Logix 5000 Advanced Process Control and Drives Instructions

1756 ControlLogix, 1756 GuardLogix, 1769 CompactLogix, 1769 Compact GuardLogix, 1789 SoftLogix, 5069 CompactLogix, 5069 Compact GuardLogix

Rockwell Automation

Original Instructions

Important User Information

Read this document and the documents listed in the additional resources section about installation, configuration, and operation of this equipment before you install, configure, operate, or maintain this product. Users are required to familiarize themselves with installation and wiring instructions in addition to requirements of all applicable codes, laws, and standards.

Activities including installation, adjustments, putting into service, use, assembly, disassembly, and maintenance are required to be carried out by suitably trained personnel in accordance with applicable code of practice.

If this equipment is used in a manner not specified by the manufacturer, the protection provided by the equipment may be impaired.

In no event will Rockwell Automation, Inc. be responsible or liable for indirect or consequential damages resulting from the use or application of this equipment.

The examples and diagrams in this manual are included solely for illustrative purposes. Because of the many variables and requirements associated with any particular installation, Rockwell Automation, Inc. cannot assume responsibility or liability for actual use based on the examples and diagrams.

No patent liability is assumed by Rockwell Automation, Inc. with respect to use of information, circuits, equipment, or software described in this manual.

Reproduction of the contents of this manual, in whole or in part, without written permission of Rockwell Automation, Inc., is prohibited.

Throughout this manual, when necessary, we use notes to make you aware of safety considerations.

WARNING: Identifies information about practices or circumstances that can cause an explosion in a hazardous environment, which may lead to personal injury or death, property damage, or economic loss.

ATTENTION: Identifies information about practices or circumstances that can lead to personal injury or death, property damage, or economic loss. Attentions help you identify a hazard, avoid a hazard, and recognize the consequence.

IMPORTANT: Identifies information that is critical for successful application and understanding of the product.

These labels may also be on or inside the equipment to provide specific precautions.

SHOCK HAZARD: Labels may be on or inside the equipment, for example, a drive or motor, to alert people that dangerous voltage may be present.

BURN HAZARD: Labels may be on or inside the equipment, for example, a drive or motor, to alert people that surfaces may reach dangerous temperatures.

ARC FLASH HAZARD: Labels may be on or inside the equipment, for example, a motor control center, to alert people to potential Arc Flash. Arc Flash will cause severe injury or death. Wear proper Personal Protective Equipment (PPE). Follow ALL Regulatory requirements for safe work practices and for Personal Protective Equipment (PPE).

The following icon may appear in the text of this document.

Identifies information that is useful and can help to make a process easier to do or easier to understand.

Rockwell Automation recognizes that some of the terms that are currently used in our industry and in this publication are not in alignment with the movement toward inclusive language in technology. We are proactively collaborating with industry peers to find alternatives to such terms and making changes to our products and content. Please excuse the use of such terms in our content while we implement these changes.

Summary of changes

This manual includes new and updated information. Use these reference tables to locate changed information.

Global changes

None for this release.

New or enhanced features

Subject	Reason
Throughout	Added references to the ControlLogix 5590 controller. The Logix Designer application versions 38 and later support it.
Throughout	Removed references to the CompactLogix 5480 controller. The Logix Designer application versions 38 and later do not support it.
Function Generator (FGEN) on page 53	In Logix Designer versions 38 and later, the Function Generator (FGEN) instruction: • is supported in the Ladder editor. • can be used in safety applications.
Enhanced PID (PIDE) on page 66	The Enhanced PID (PIDE) instruction is not supported in safety applications and is not supported on ControlLogix 5590 controllers. For ControlLogix 5590 controller projects, substitute the PPID instruction. See Replacement Guidelines: Logix 5000 Controllers (publication 1756-RM100) for information on migrating projects for use with ControlLogix 5590 controllers.

Preface

This manual provides a programmer with details about the available General, Motion, Process, and Drives instruction set for a Logix-based controller.

If you design, program, or troubleshoot safety applications that use GuardLogix controllers, refer to the GuardLogix Safety Application Instruction Set Safety Reference Manual, publication 1756-RM095.

This manual is one of a set of related manuals that show common procedures for programming and operating Logix 5000™ controllers.

For a complete list of common procedures manuals, refer to the Logix 5000 Controllers Common Procedures Programming Manual, publication 1756-PM001.

The term Logix 5000 controller refers to any controller that is based on the Logix 5000 operating system.

Studio 5000 environment

The Studio 5000 Automation Engineering & Design Environment® combines engineering and design elements into a common environment. The first element is the Studio 5000 Logix Designer® application. The Logix Designer application is the rebranding of RSLogix 5000® software and will continue to be the product to program Logix 5000™ controllers for discrete, process, batch, motion, safety, and drive-based solutions.

The Studio 5000® environment is the foundation for the future of Rockwell Automation® engineering design tools and capabilities. The Studio 5000 environment is the one place for design engineers to develop all elements of their control system.

Find instructions in manuals

Refer to these tables to find the applicable Logix5000 controllers instruction manual for each instruction.

Table 1. Logix5000 Controllers General Instructions Reference Manual (publication 1756-RM018)

Absolute Value (ABS)	Compute (CPT)	Less Than (LT)	Cosine (COS)

Table 1. Logix5000 Controllers General Instructions Reference Manual (publication 1756-RM018) (continued)

Add (ADD)	Count down (CTD)	Less Than or Equal to (LE)	Jump to Subroutine (JSR), Subroutine (SBR), and Return (RET)
Analog Alarm (ALMA)	Count up (CTU)	LIFO Load (LFL)	Reset SFC (SFR)
Always False (AFI)	Count up/down CTUD	LIFO Unload (LFU)	Return (RET)
Arc Cosine (ACOS)	Data Transition (DTR)	License Validation (LV)	Retentive Timer On (RTO)
Arc Sine (ASIN)	Degrees (DEG)	Limit (LIMIT)	Retentive Timer On with Reset (RTOR)
Arc Tangent (ATAN)	Diagnostic Detect (DDT)	Log Base (LOG)	Pause SFC (SFP)
ASCII Chars in Buffer (ACB)	Digital Alarm (ALMD)	Lower to Case (LOWER)	Size In Elements (SIZE)
ASCII Clear Buffer (ACL)	DINT To String (DTOS)	Masked Move (MVM)	Sequencer Input (SQI)
ASCII Handshake Lines (AHL)	Divide (DIV)	Masked Move with Target (MVMT)	Sequencer Load (SQL)
ASCII Read (ARD)	End of Transition (EOT)	Master Control Reset (MCR)	Sequencer Output (SQO)
ASCII Read Line (ARL)	Equal to (EQ)	Masked Equal to (MEQ)	Sine (SIN)
ASCII Test for Buffer Line (ABL)	Examine if Closed (XIC)	Message (MSG)	Square Roost (SQRT)
ASCII Write (AWT)	Examine If Open (XIO)	Middle String (MID)	String Concatenate (CONCAT)
ASCII Write Append (AWA)	File Arithmetic (FAL)	Modulo (MOD)	String Delete (DELETE)
Bit Field Distribute (BTD)	File Bit Comparison (FBC)	Move (MOVE)	String to DINT (STOD)
Bit Field Distribute with Target (BTDT)	FIFO Load (FFL)	Multiply (MUL)	String to REAL (STOR)
Bit Shift Left (BSL)	FIFO Unload (FFU)	Natural Log (LN)	Swap Byte (SWPB)
Bit Shift Right (BSR)	File Average (AVE)	Negate (NEG)	Subtract (SUB)
Bitwise And (AND)	File Standard Deviation (STD)	Not Equal to (NE)	Tangent (TAN)
Bitwise (NOT)	File Fill (FLL)	No Operation (NOP)	Timer Off Delay (TOF)
Bitwise (OR)	File Sort (SRT)	One Shot (ONS)	Timer Off Delay with Reset (TOFR)
Bitwise Exclusive (XOR)	Find String (FIND)	One Shot Falling (OSF)	Timer On Delay (TON)
Boolean AND (BAND)	For (FOR)	One Shot Falling with Input (OSFI)	Timer On Delay with Reset (TONR)
Boolean Exclusive OR (BXOR)	File Search and Compare (FSC)	One Shot Rising (OSR)	Temporary End (TND)
Boolean NOT (BNOT)	Get System Value (GSV) and Set System Value (SST)	One Shot Rising with Input (OSRI)	Trigger Event Task (EVENT)
Boolean OR (BOR)	Greater Than or Equal to (GE)	Output Energize (OTE)	Truncate (TRUNC)

Table 1. Logix5000 Controllers General Instructions Reference Manual (publication 1756-RM018) (continued)

Break (BRK)	Greater than (GT)	Output Latch (OTL)	Unknown Instruction (UNK)
Clear (CLR)	Insert String (INSERT)	Output Unlatch (OTU)	Upper Case (UPPER)
Compare (CMP)	Immediate Output (IOT)	Proportional Integral Derivative (PID)	User Interrupt Disable (UID)/User Interrupt Enable (UIE)
Convert to BCD (TO_BCD)	Is Infinity (IsINF)	Radian (RAD)	X to the Power of Y (EXPT)
Convert to Integer (BCD_TO)	Is Not a Number (IsNAN)	Real to String (RTOS)	
Copy File (COP), Synchronous Copy File (CPS)	Jump to Label (JMP) and Label (LBL)	Reset (RES)	

Table 2. Logix5000 Controllers Advanced Process Control, Drives, Equipment Phase, and Sequence Instructions Reference Manual (publication 1756-RM006)

Alarm (ALM)	Equipment Phase External Request (PXRQ)	JK Flip-Flop (JKFF)	Rate Limiter (RLIM)
Attach to Equipment Phase (PATT)	Equipment Phase Failure (PFL)	Lead-Lag (LDLG)	Reset Dominant (RESD)
Attach to Equipment Sequence (SATT)	Equipment Phase New Parameters (PRNP)	Low Pass Filter (LPF)	Scale (SCL)
Coordinated Control (CC)	Equipment Phase Override Command (POVR)	Maximum Capture (MAXC)	S-Curve (SCRV)
D Flip-Flop (DFF)	Equipment Phase Paused (PPD)	Minimum Capture (MINC)	Second-Order Controller (SOC)
Deadtime (DEDT)	Equipment Sequence Assign Sequence Identifier (SASI)	Modular Multivariable Control (MMC)	Second-Order Lead Lag (LDL2)
Derivative (DERV)	Equipment Sequence Clear Failure (SCLF)	Moving Average (MAVE)	Select (SEL)
Detach from Equipment Phase (PDET)	Equipment Sequence command (SCMD)	Moving Standard Deviation (MSTD)	Selected Negate (SNEG)
Detach from Equipment Sequence (SDET)	Equipment Sequence Override (SOVR)	Multiplexer (MUX)	Selected Summer (SSUM)
Discrete 3-State Device (D3SD)	Function Generator (FGEN)	Notch Filter (NTCH)	Set Dominant (SETD)
Discrete 2-State Device (D2SD)	High Pass Filter (HPF)	Phase State Complete (PSC)	Split Range Time Proportional (SRTP)
Enhanced PID (PIDE)	High/Low Limit (HLL)	Position Proportional (POSP)	Totalizer (TOT)
Enhanced Select (ESEL)	HMI Button Control (HMIBC)	Proportional + Integral (PI)	Up/Down Accumulator (UPDN)
Equipment Phase Clear Failure (PCLF)	Integrator (INTG)	Pulse Multiplier (PMUL)	

Table 2. Logix5000 Controllers Advanced Process Control, Drives, Equipment Phase, and Sequence Instructions Reference Manual (publication 1756-RM006) (continued)

Equipment Phase Command (PCMD)	Internal Model Control (IMC)	Ramp/Soak (RMPS)	

Table 3. Logix5000 Controllers Motion Instructions Reference Manual (publication MOTION-RM002)

Master Driven Coordinated Control (MDCC)	Motion Axis Stop (MAS)	Motion Coordinated Circular Move (MCCM)	Motion Group Shutdown (MGSD)
Motion Apply Axis Tuning (MAAT)	Motion Axis Time Cam (MATC)	Motion Coordinated Linear Move (MCLM)	Motion Group Shutdown Reset (MGSR)
Motion Apply Hookup Diagnostics (MAHD)	Motion Axis Shutdown (MASD)	Motion Coordinated Shutdown (MCSD)	Motion Group Stop (MGS)
Motion Arm Output Cam (MAOC)	Motion Axis Shutdown Reset (MASR)	Motion Coordinated Shutdown Reset (MCSR)	Motion Group Strobe Position (MGSP)
Motion Arm Registration (MAR)	Motion Calculate Cam Profile (MCCP)	Motion Coordinated Stop (MCS)	Motion Redefine Position (MRP)
Motion Arm Watch (MAW)	Motion Coordinated Path Move (MCPM)	Motion Coordinated Transform (MCT)	Motion Run Axis Tuning (MRAT)
Motion Axis Fault Reset (MAFR)	Motion Calculate Slave Values (MCSV)	Motion Direct Drive Off (MDF)	Motion Run Hookup Diagnostics (MRHD)
Motion Axis Gear (MAG)	Motion Coordinated Transform with Orientation (MCTO)	Motion Direct Drive On (MDO)	Motion Servo Off (MSF)
Motion Axis Home (MAH)	Motion Calculate Transform Position (MCTP)	Motion Direct Start (MDS)	Motion Servo On (MSO)
Motion Axis Jog (MAJ)	Motion Calculate Transform Position with Orientation (MCTPO)	Motion Disarm Output Cam (MDOC)	
Motion Axis Move (MAM)	Motion Change Dynamics (MCD)	Motion Disarm Registration (MDR)	
Motion Axis Position Cam (MAPC)	Motion Coordinated Change Dynamics (MCCD)	Motion Disarm Watch (MDW)	

Table 4. Logix 5000 Controller Safety Application Instruction Set (publication 1756-

Auxiliary Valve Control (AVC)	Dual Channel Input Stop (DCS)	Light Curtain (LC)	Safely-Limited Position (SLP)
CamShaft Monitor (CSM)	Dual Channel Input Stop with Test and Lock (DCSTL)	Main Valve Control (MVC)	Safely-Limited Speed (SLS)
Clutch Brake Continuous Mode (CBCM)	Dual Channel Input Stop with Test and Mute (DCSTM)	Maintenance Manual Valve Control (MMVC)	Safety Feedback Interface (SFX)
Clutch Brake Inch Mode (CBIM)	Dual-channel Input Start (DCSRT)	Redundant Input (RIN)	Safety Mat (SMAT)
Clutch Brake Single Stroke Mode (CBSSM)	Dual-Channel Input Stop with Test (DCST)	Redundant Output (ROUT)	Two Hand Run Station (THRS)

Table 4. Logix 5000 Controller Safety Application Instruction Set (publication 1756- (continued)

Configurable Redundant Output (CROUT)	Eight Position Mode Selector (EPMS)	Safe Brake Control (SBC)	Two Hand Run Station Enhanced (THRSe)
Crankshaft Position Monitor (CPM)	Emergency Stop (ESTOP)	Safe Direction (SDI)	Two Sensor Asymmetrical Muting (TSAM)
Diverse Input (DIN)	Enable Pendant (ENPEN)	Safe Operating Stop (SOS)	Two-sensor Symmetrical Muting (TSSM)
Dual Channel Analog Input (DCA - integer version) and (DCAF - floating point version)	Five Position Mode Selector (FPMS)	Safe Stop 1 (SS1)	
Dual Channel Input Monitor (DCM)	Four Sensor Bi-Directional Muting (FSBM)	Safe Stop 2 (SS2)	

Table 5. PlantPAx Process Control Instructions (publication PROCES-RM215)

Process Analog HART (PAH)	Process Discrete 2-, 3-, or 4-State Device (PD4SD)	Process Interlocks (PINTLK)	Process Run Time and Start Counter (PRT)
Process Analog Input (PAI)	Process Deadband Controller (PDBC)	Process Lead Lag Standby Motor Group (PLLS)	Process Tank Strapping Table (PTST)
Process Dual Sensor Analog Input (PAID)	Process Discrete Input (PDI)	Process Motor (PMTR)	Process Valve (PVLV)
Process Multi Sensor Analog Input (PAIM)	Process Discrete Output (PDO)	Process Permissives (PPERM)	Process Valve Statistics (PVLVS)
Process Analog Output (PAO)	Process Dosing (PDOSE)	Process Proportional + Integral + Derivative (PPID)	
Process Boolean Logic (PBL)	Process Analog Fanout (PFO)	Process Pressure/Temperature Compensated Flow (PPTC)	
Process Command Source (PCMDSRC)	Process High or Low Selector (PHLS)	Process Restart Inhibit (PRI)	

Additional resources

These documents contain additional information concerning related Rockwell Automation products.

Resource	Description
Industrial Automation Wiring and Grounding Guidelines, publication, 1770-4.1	Provides general guidelines for installing a Rockwell Automation industrial system.
Rockwell Automation product certifications	Provides declarations of conformity, certificates, and other certification details.

View or download publications at https://www.rockwellautomation.com/en-us/support/documentation/literature-library.html. To order paper copies of technical documentation, contact a local Rockwell Automation distributor or sales representative.

Legal notices

Rockwell Automation publishes legal notices, such as privacy policies, license agreements, trademark disclosures, and other terms and conditions on the Legal Notices page of the Rockwell Automation website.

Software and Cloud Services Agreement

Review the Rockwell Automation Software and Cloud Services Agreement here.

Open Source Software Licenses

The software included in this product contains copyrighted software that is licensed under one or more open source licenses.

You can view a full list of all open source software used in this product and their corresponding licenses at this URL:

Studio 5000 Logix Designer Open Source Attribution List

You may obtain Corresponding Source code for open source packages included in this product from their respective project web site(s). Alternatively, you may obtain complete Corresponding Source code by contacting Rockwell Automation via the **Contact** form on the Rockwell Automation website: http://www.rockwellautomation.com/global/about-us/contact/contact.page. Please include "Open Source" as part of the request text.

Process Control Instructions

The Process Control instructions include these instructions:

Available Instructions

Ladder Diagram

Not available

Function Block and Structured Text

ALM on page 15	SCL on page 134	PIDE on page 66	RMPS on page 117	POSP on page 109	SRTP on page 138	LDLG on page 61	FGEN on page 53

TOT on page 146	DEDT on page 48	D2SD on page 37	D3SD on page 20	IMC on page 208	CC on page 155	MMC on page 236

If you want to	Use this instruction
Provide alarming for any analog signal.	ALM
Control discrete devices, such as solenoid valves, pumps, and motors, that have only two possible states (e.g., on/off, open/closed, etc.).	D2SD
Control discrete devices, such as high/low/off feeders that have three possible states (e.g., fast/slow/off, forward/stop/reverse, etc.).	D3SD
Perform a delay of a single input. You select the amount of deadtime delay.	DEDT
Convert an input based on a piece-wise linear function.	FGEN
Provide a phase lead-lag compensation for an input signal.	LDLG
Regulate an analog output to maintain a process variable at a certain setpoint, using a PID algorithm.	PIDE
Raise/lower or open/close a device, such as a motor-operated valve, by pulsing open or close contacts.	POSP
Provide for alternating ramp and soak periods to follow a temperature profile.	RMPS
Convert an unscaled input value to a floating point value in engineering units.	SCL

Take the 0-100% output of a PID loop and drive heating and cooling digital output contacts with a periodic pulse.	SRTP
Provide a time-scaled accumulation of an analog input value, such as a volumetric flow.	TOT
Control a single process variable by maintaining a single controller output.	IMC
Control a single process variable by manipulating as many as three different control variables.	CC
Control two process variables to their setpoints using up to three control variables.	MMC

Alarm (ALM)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, Compact GuardLogix 5380, CompactLogix 5380, ControlLogix 5580, GuardLogix 5580, and ControlLogix 5590 controllers.

The Alarm (ALM) instruction provides alarming for any analog signal.

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram logic.

Function Block

Structured Text

ALM(ALM_tag)

Operands

Function Block

Operand	Type	Format	Description

ALM tag	ALARM	structure	ALM structure

Structured Text

Operand	Type	Format	Description
ALM tag	ALARM	structure	ALM structure

See *Structured Text Syntax* for more information on the syntax of expressions within structured text.

ALARM Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If false, the instruction does not execute and outputs are not updated. Default is true.
In	REAL	The analog signal input. Valid = any float Default = 0.0
HHLimit	REAL	The high-high alarm limit for the input. Valid = any real value Default = maximum positive value
HLimit	REAL	The high alarm limit for the input. Valid = any real value Default = maximum positive value
LLimit	REAL	The low alarm limit for the input. Valid = any real value Default = maximum negative value
LLLimit	REAL	The low-low alarm limit for the input. Valid = any real value Default = maximum negative value
≤ Deadband	REAL	The alarm deadband for the high-high to low-low limits Valid = any real value ≥ 0.0 Default = 0.0
ROCPosLimit	REAL	The rate-of-change alarm limit in units per second for a positive (increasing) change in the input. Set ROCPosLimit = 0 to disable ROC positive alarming. If invalid, the instruction assumes a value of 0.0 and sets the appropriate bit in Status.

Input Parameter	Data Type	Description
		Valid = any real value ≤ 0.0 Default = 0.0
ROCNegLimit	REAL	The rate-of-change alarm limit in units per second for a negative (decreasing) change in the input. Set ROCNegLimit = 0 to disable ROC negative alarming. If invalid, the instruction assumes a value of 0.0 and sets the appropriate bit in Status. Valid = any real value ≤ 0.0 Default = 0.0
ROCPeriod	REAL	Time period in seconds for calculation (sampling interval) of the rate of change value. Each time the sampling interval expires, a new sample of In is stored, and ROC is re-calculated. Instead of an enable bit like other conditions in the analog alarm, the rate-of-change detection is enabled by putting any non-zero value in the ROCPeriod. Valid = 0.0 to 32767.0 Default = 0.0.

Output Parameter	Data Type	Description
EnableOut	BOOL	Indicates if instruction is enabled. Cleared to false if ROC overflows.
HHAlarm	BOOL	The high-high alarm indicator. Default = false
HAlarm	BOOL	The high alarm indicator. Default = false
LAlarm	BOOL	The low alarm indicator. Default = false
LLAlarm	BOOL	The low-low alarm indicator. Default = false
ROCPosAlarm	BOOL	The rate-of-change positive alarm indicator. Default = false
ROCNegAlarm	BOOL	The rate-of-change negative alarm indicator.

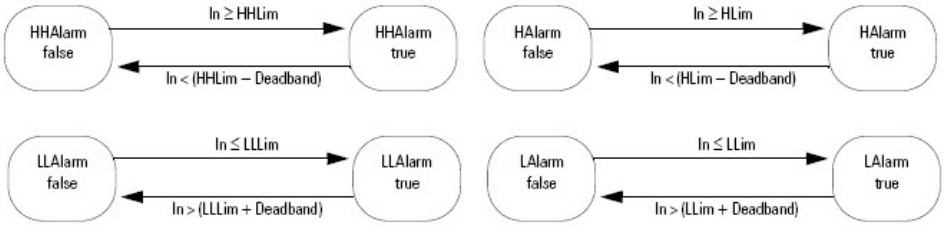
Output Parameter	Data Type	Description
		Default = false
ROC	REAL	The rate-of-change output.
Status	DINT	Status of the function block.
InstructFault (Status.0)	BOOL	The instruction detected one of the following execution errors. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.
DeadbandInv (Status.1)	BOOL	Invalid Deadband value.
ROCPosLimitInv (Status.2)	BOOL	Invalid ROCPosLimit value.
ROCNegLimitInv (Status.3)	BOOL	Invalid ROCNegLimit value.
ROCPeriodInv (Status.4)	BOOL	Invalid ROCPeriod value.

Description

The ALM instruction provides alarm indicators for high-high, high, low, low-low, rate-of-change positive, and rate-of-change negative. An alarm deadband is available for the high-high to low-low alarms. A user-defined period for performing rate-of-change alarming is also available.

High-high to Low-low Alarm

The high-high and low-low alarm algorithms compare the input to the alarm limit and the alarm limit plus or minus the deadband.



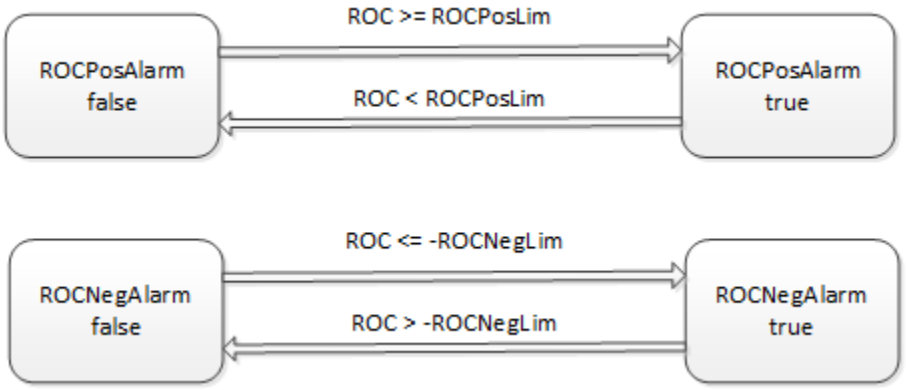
Rate-of-change Alarm

The rate-of-change (ROC) alarm compares the change of the input over the ROCPeriod to the rate-of-change limits. The ROCPeriod provides a type of deadband for the rate-of-change alarm. For example, define an ROC alarm limit of 2^0F/second with a period of execution of 100 ms. If you use an analog input module with a resolution of 1^0F, every time the input value changes, an ROC alarm is generated because the instruction calculates an effective rate of 10°F/second. However, enter an ROCPeriod of 1 sec and the instruction only generates an alarm if the rate truly exceeds the 2^0F/second limit.

The ROC alarm calculates the rate-of-change as:

$$ROC = \frac{In(Now) - In(EndofpreviousROCPeriod)}{ROCPeriod}$$

The instruction performs this calculation when the ROCPeriod expires. Once the instruction calculates the ROC, it determines alarms as:



Monitoring the ALM Instruction

There is an operator faceplate available for the ALM instruction.

Affects Math Status Flags

No

Major/Minor Faults

None specific to this instruction. See *Common Attributes* for operand-related faults.

Execution

Function Block

Condition/State	Action Taken
Prescan	Rung-condition-in bits are cleared to false.
Rung-condition-in is false	Rung-condition-in bits are cleared to false.
Rung-condition-in is true	Rung-condition-in bits are set to true. The instruction executes.
Postscan	Rung-condition-in bits are cleared to false.

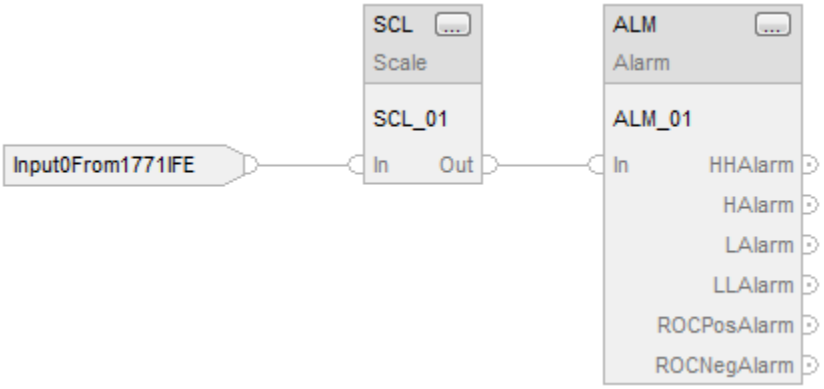
Structured Text

Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.
Normal Execution	See Rung-condition-in is true in the Function Block table.
Postscan	See Postscan in the Function Block table.

Example

The ALM instruction is typically used either with analog input modules (such as 1771 I/O modules) that do not support on-board alarming, or to generate alarms on a calculated variable. In this example, an analog input from a 1771-IFE module is first scaled to engineering units using the SCL instruction. The Out of the SCL instruction is an input to the ALM instruction to determine whether to set an alarm. The resulting alarm output parameters could then be used in your program and/or viewed on an operator interface display.

Function Block



Structured Text

```
SCL_01.IN := Input0From1771IFE;

SCL(SCL_01);

ALM_01.IN := SCL_01.Out;

ALM(ALM_01);
```

Discrete 3-State Device (D3SD)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, Compact GuardLogix 5380, CompactLogix 5380, ControlLogix 5580, GuardLogix 5580, and ControlLogix 5590 controllers.

The Discrete 3-State Device (D3SD) instruction controls a discrete device having three possible states, such as fast/slow/off or forward/stop/reverse.

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram logic.

Function Block

Structured Text

D3SD(D3SD_tag)

Operands

Structured Text

Operand	Type	Format	Description
D3SD tag	DISCRETE_3STATE	structure	D3SD structure

See *Structured Text Syntax* for more information on the syntax of expressions within structured text.

Function Block

Operand	Type	Format	Description
D3SD tag	DISCRETE_3STATE	structure	D3SD structure

DISCRETE_3STATE Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If false, the instruction does not execute and outputs are not updated. Default is true.
Prog0Command	BOOL	Program state 0 command. This input determines the device state when the device is in Program control. If true, the device is commanded to the 0 state. Default is false.
Prog1Command	BOOL	Program state 1 command. This input determines the device state when the device is in Program control. If true, the device is commanded to the 1 state. Default is false.
Prog2Command	BOOL	Program state 2 command. This input determines the device state when the device is in Program control. If true, the device is commanded to the 2 state. Default is false.
Oper0Req	BOOL	Operator state 0 request. Set to true by the operator interface to place the device into the 0 state when the device is in Operator control. Default is false.
Oper1Req	BOOL	Operator state 1 request. Set true by the operator interface to place the device into the 1 state when the device is in Operator control. Default is false.
Oper2Req	BOOL	Operator state 2 request. Set to true by the operator interface to place the device into the 2 state when the device is in Operator control. Default is false.
State0Perm	BOOL	State 0 permissive. Unless in Hand or Override mode, this input must be true for the device to enter the 0 state. This input has no effect if the device is already in the 0 state. Default is true.

Input Parameter	Data Type	Description
State1Perm	BOOL	State 1 permissive. Unless in Hand or Override mode, this input must be true for the device to enter the 1 state. This input has no effect if the device is already in the 1 state. Default is true.
State2Perm	BOOL	State 2 permissive. Unless in Hand or Override mode, this input must be true for the device to enter the 2 state. This input has no effect if the device is already in the 2 state. Default is true.
FB0	BOOL	The first feedback input available to the instruction. Default is false.
FB1	BOOL	The second feedback input available to the instruction. Default is false.
FB2	BOOL	The third feedback input available to the instruction. Default is false.
FB3	BOOL	The fourth feedback input available to the instruction. Default is false.
HandFB0	BOOL	Hand feedback state 0. This input from a field hand/off/auto station shows the requested state of the field device. True indicates the field device is being requested to enter the 0 state; false indicates the field device is being requested to enter some other state. Default is false.
HandFB1	BOOL	Hand feedback state 1. This input from a field hand/off/auto station shows the requested state of the field device. True indicates the field device is being requested to enter the 1 state; false indicates the field device is being requested to enter some other state. Default is false.
HandFB2	BOOL	Hand feedback state 2. This input from a field hand/off/auto station shows the requested state of the field device. True indicates the field

Input Parameter	Data Type	Description
		device is being requested to enter the 2 state; false indicates the field device is being requested to enter some other state. Default is false.
FaultTime	REAL	Fault time value. Configure the value in seconds of the time to allow the device to reach a newly commanded state. Set FaultTime = 0 to disable the fault timer. If this value is invalid, the instruction assumes a value of zero and sets the appropriate bit in Status. Valid = any float ≥ 0.0 Default = 0.0
FaultAlarmLatch	BOOL	Fault alarm latch input. When true and FaultAlarm is true, latch FaultAlarm. To unlatch FaultAlarm, set FaultAlmUnlatch to true or clear FaultAlarmLatch to false. Default is false.
FaultAlmUnLatch	BOOL	Fault alarm unlatch input. Set this input to true when FaultAlarmLatch is set to unlatch FaultAlarm. The instruction clears this input to false. Default is false.
OverrideOnInit	BOOL	Override on initialization request. If this bit is true, then during instruction first scan, the instruction is placed in Operator control with Override true and Hand false. If ProgHandReq is true, then Override is cleared to false and Hand is set to true. Default is false.
OverrideOnFault	BOOL	Override on fault request. Set this value to true if the device should go to Override mode and enter the Override State on a fault alarm. After the fault alarm is removed, the instruction is placed in Operator control. Default is false.
Out0State0	BOOL	Output 0 state 0 input. This value determines the value of Output0 when the device is in the 0 state. Default is false.

Input Parameter	Data Type	Description
Out0State1	BOOL	Output 0 state 1 input. This value determines the value of Output0 when the device is in the 1 state. Default is false.
Out0State2	BOOL	Output 0 state 2 input. This value determines the value of Output0 when the device is in the 2 state. Default is false.
Out1State0	BOOL	Output 1 state 0 input. This value determines the value of Output1 when the device is in the 0 state. Default is false.
Out1State1	BOOL	Output 1 state 1 input. This value determines the value of Output1 when the device is in the 1 state. Default is false.
Out1State2	BOOL	Output 1 state 2 input. This value determines the value of Output1 when the device is in the 2 state. Default is false.
Out2State0	BOOL	Output 2 state 0 input. This value determines the value of Output2 when the device is in the 0 state. Default is false.
Out2State1	BOOL	Output 2 state 1 input. This value determines the value of Output2 when the device is in the 1 state. Default is false.
Out2State2	BOOL	Output 2 state 2 input. This value determines the value of Output2 when the device is in the 2 state. Default is false.
OverrideState	DINT	Override state input. Set this input to indicate the state of the device when in Override mode. 2 = Device should go to the 2 state 1 = Device should go to the 1 state 0 = Device should go to the 0 state An invalid value sets the appropriate bit in Status. Valid = 0 to 2 Default = 0
FB0State0	BOOL	Feedback 0 state 0 input. This value determines the expected value of

Input Parameter	Data Type	Description
		FB0 when the device is in the 0 state. Default is false.
FB0State1	BOOL	Feedback 0 state 1 input. This value determines the expected value of FB0 when the device is in the 1 state. Default is false.
FB0State2	BOOL	Feedback 0 state 2 input. This value determines the expected value of FB0 when the device is in the 2 state. Default is false.
FB1State0	BOOL	Feedback 1 state 0 input. This value determines the expected value of FB1 when the device is in the 0 state. Default is false.
FB1State1	BOOL	Feedback 1 state 1 input. This value determines the expected value of FB1 when the device is in the 1 state. Default is false.
FB1State2	BOOL	Feedback 1 state 2 input. This value determines the expected value of FB1 when the device is in the 2 state. Default is false.
FB2State0	BOOL	Feedback 2 state 0 input. This value determines the expected value of FB2 when the device is in the 0 state. Default is false.
FB2State1	BOOL	Feedback 2 state 1 input. This value determines the expected value of FB2 when the device is in the 1 state. Default is false.
FB2State2	BOOL	Feedback 2 state 2 input. This value determines the expected value of FB2 when the device is in the 2 state. Default is false.
FB3State0	BOOL	Feedback 3 state 0 input. This value determines the expected value of FB3 when the device is in the 0 state. Default is false.
FB3State1	BOOL	Feedback 3 state 1 input. This value determines the expected value of FB3 when the device is in the 1 state. Default is false.

Input Parameter	Data Type	Description
FB3State2	BOOL	Feedback 3 state 2 input. This value determines the expected value of FB3 when the device is in the 2 state. Default is false.
ProgProgReq	BOOL	Program program request. Set to true by the user program to request Program control. Ignored if ProgOperReq is true. Holding this true and ProgOperReq false locks the instruction in Program control. Default is false.
ProgOperReq	BOOL	Program operator request. Set to true by the user program to request operator control. Holding this true locks the instruction in Operator control. Default is false.
ProgOverrideReq	BOOL	Program override request. Set to true by the user program to request the device to enter Override mode. Ignored if ProgHandReq is true. Default is false.
ProgHandReq	BOOL	Program hand request. Set to true by the user program to request the device to enter Hand mode. Default is false.
OperProgReq	BOOL	Operator program request. Set to true by the operator interface to request Program control. The instruction clears this input to false. Default is false.
OperOperReq	BOOL	Operator operator request. Set to true by the operator interface to request Operator control. The instruction clears this input to false. Default is false.
ProgValueReset	BOOL	Reset program control values. When true, all the program request inputs are cleared to false at each execution of the instruction. Default is false.

Output Parameter	Data Type	Description
EnableOut	BOOL	Indicates if instruction is enabled.

Output Parameter	Data Type	Description
Out0	BOOL	The first output of the instruction.
Out1	BOOL	The second output of the instruction.
Out2	BOOL	The third output of the instruction.
Device0State	BOOL	Device state 0 output. True when the device is commanded to the 0 state and the feedback indicates the device really is in the 0 state.
Device1State	BOOL	Device state 1 output. True when the device is commanded to the 1 state and the feedback indicates the device really is in the 1 state.
Device2State	BOOL	Device state 2 output. True when the device is commanded to the 2 state and the feedback indicates the device really is in the 2 state.
Command0Status	BOOL	Device state 0 command status. True when the device is being commanded to the 0 state; false when the device is being commanded to some other state.
Command1Status	BOOL	Device state 1 command status. True when the device is being commanded to the 1 state; false when the device is being commanded to some other state.
Command2Status	BOOL	Device state 2 command status. True when the device is being commanded to the 2 state; false when the device is being commanded to some other state.
FaultAlarm	BOOL	Fault alarm output. True if the device has been commanded to a new state, and the FaultTime has expired without the feedback indicating that the new state has actually been reached. Also set to true if, after reaching a commanded state, the feedbacks suddenly indicate that the device is no longer in the commanded state.
ModeAlarm	BOOL	Mode alarm output. True if the device is in operator control and a

Output Parameter	Data Type	Description
		ProgxCommand input requests a state which is different from the state currently commanded by the operator. This alarm is intended as a reminder that a device was left in Operator control.
ProgOper	BOOL	Program/operator control indicator. True when in Program control. False when in Operator control.
Override	BOOL	Override mode. True when the device is in the Override mode.
Hand	BOOL	Hand mode. True when the device is in the Hand mode.
Status	DINT	Status of the function block.
InstructFault (Status.0)	BOOL	The instruction detected one of the following execution errors. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.
FaultTimeInv (Status.1)	BOOL	Invalid FaultTime value. The instruction sets FaultTime = 0.
OverrideStateInv (Status.2)	BOOL	The Override value is out of range. It prevents the instruction from entering the Override state.
ProgCommandInv (Status.3)	BOOL	Multiple program state command bits are set at the same time. Refer to Commanded State in Program Control section.
OperReqInv (Status.4)	BOOL	Multiple operator state request bits are set at the same time. Refer to Commanded State in Program Control section.
HandCommandInv (Status.5)	BOOL	Multiple hand feedback state request bits are set at the same time.

Description

The D3SD instruction controls a discrete device having three possible states, such as fast/slow/off or forward/stop/reverse. Typical discrete devices of this nature include feeder systems, and reversible motors.

Affects Math Status Flags

No

Major/Minor Faults

None specific to this instruction. See Common Attributes on page for operand-related faults.

Execution

Function Block

Condition/State	Action Taken
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is true	EnableIn and EnableOut bits are set to true. The instruction executes
Instruction first run	Set ProgOper to Operator Mode. Set Command0Status to True. Set Command1Status to False. Set Command2Status to False.
Instruction first scan	The fault timer is cleared. ModeAlarm is cleared to false. All the operator request inputs are cleared to false. If ProgValueReset is true, all the program request inputs are cleared to false. When OverrideOnInit is true, ProgOper is cleared to false(Operator control). If ProgHandReq is false and OverrideOnInit is true, Hand is cleared to false and Override is set to true (Override mode). If ProgHandReq is true, Hand is set to true and Override is cleared to false(Hand mode).
Postscan	EnableIn and EnableOut bits are cleared to false.

Structured Text

In Structured Text, EnableIn is always true during normal scan. Therefore, if the instruction is in the control path activated by the logic it will execute.

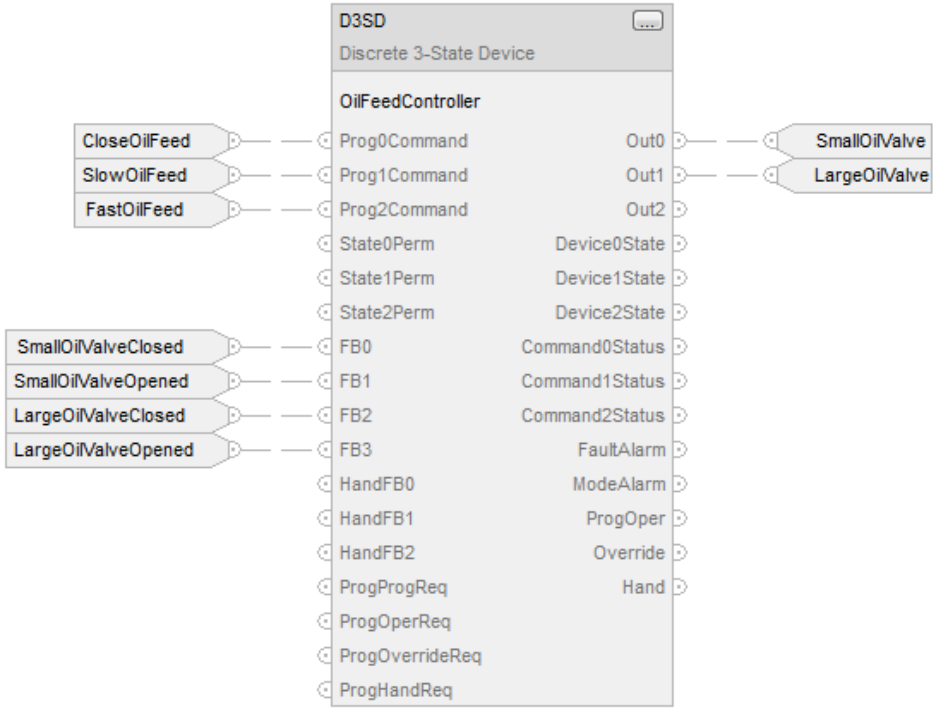
Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.

Condition/State	Action Taken
Normal Execution	See Tag.EnableIn is true in the Function Block table.
Postscan	See Postscan in the Function Block table.

Examples

The D3SD instruction is typically used to control 3-state devices such as high/low/off feed systems. In this example, the D3SD instruction controls a feed system consisting of a pair of solenoid valves adding vegetable oil to a batch tank. One of the valves is on a large diameter feed pipe into the batch tank, and the other valve is plumbed in parallel on a small diameter feed pipe. When oil is first added, the D3SD instruction is commanded to the fast feed state (state 2) where both valves are opened. When the oil added approaches the target amount, the D3SD instruction is commanded to the slow feed state (state 1) where the "large valve" is closed and the "small valve" is kept open. When the target is reached, the D3SD instruction is commanded to go to the off state (state 0) and both valves are closed. As long as the D3SD instruction is in Program control, the valves open according to the CloseOilFeed, SlowOilFeed, and FastOilFeed inputs. The operator can also take Operator control of the feed system if necessary. The solenoid valves in this example have limit switches which indicate when the valves are fully closed or opened. These switches are wired into the FB0, FB1, FB2, and FB3 feedback inputs. This allows the D3SD instruction to generate a FaultAlarm if the solenoid valves do not reach their commanded states within the configured FaultTime.

Function Block



Structured Text

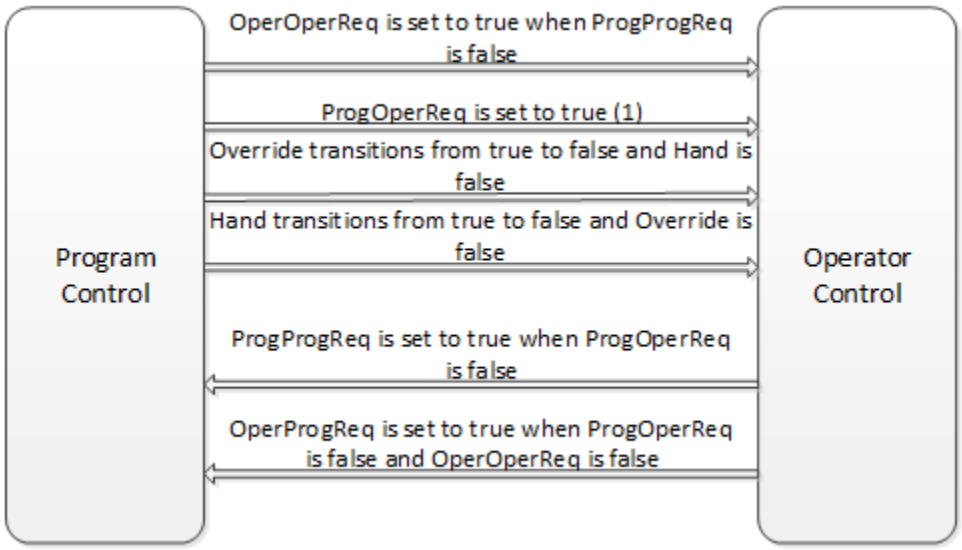
OilFeedController.Prog0Command := ClosedOilFeed;

OilFeedController.Prog1Command := SlowOilFeed;

```
OilFeedController.Prog2Command := FastOilFeed;
OilFeedController.FB0 := SmallOilValveClosed;
OilFeedController.FB1 := SmallOilValveOpened;
OilFeedController.FB2 := LargeOilValveClosed;
OilFeedController.FB3 := LargeOilValveOpened;
D3SD(OilFeedController);
SmallOilValve := OilFeedController.Out0;
LargeOilValve := OilFeedController.Out1;
```

Switch Between Program Control and Operator Control

The following diagram shows how the D3SD instruction changes between Program control and Operator control.



(1) The instruction remains in Operator control mode when ProgOperReq is true.

Commanded State in Program Control

The following table shows how the D3SD instruction operates when in Program control.

Prog0 Command	Prog1 Command	Prog2 Command	State0 Perm	State1 Perm	State2 Perm	Description
false	false	true	either	either	true	Command0Status is cleared to false Command1Status is cleared to false Command2Status is set to true

Prog0 Command	Prog1 Command	Prog2 Command	State0 Perm	State1 Perm	State2 Perm	Description
false	true	false	either	true	either	Command0Status is cleared to false Command1Status is set to true Command2Status is cleared to false
true	false	false	true	either	either	Command0Status is set to true Command1Status is cleared to false Command2Status is cleared to false

If more than one program command input is true:

- The instruction sets the appropriate bit in Status
- If Override and Hand are cleared to false, the instruction holds the previous state

Commanded State in Operator Control

The following table shows how the D3SD instruction operates when in Operator control.

Oper0Req	Oper1Req	Oper2Req	State0 Perm	State1 Perm	State2 Perm	Description
false	false	true	either	either	true	Command0Status is cleared to false Command1Status is cleared to false Command2Status is set to true
false	true	false	either	true	either	Command0Status is cleared to false Command1Status is set to true Command2Status is cleared to false

Oper0Req	Oper1Req	Oper2Req	State0 Perm	State1 Perm	State2 Perm	Description
true	false	false	true	either	either	Command0Status is set to true Command1Status is cleared to false Command2Status is cleared to false

If more than one operator command input is true:

- The instruction sets the appropriate bit in Status
- If Override and Hand are cleared to false, the instruction holds the previous state

After every instruction execution, the instruction:

- Clears all the operator request inputs
- If ProgValueReset is true, clears all the program request inputs to false

Hand Mode or Override Mode

The following table describes how the D3SD instruction determines whether to operate in Hand or Override mode.

ProgHandReq	ProgOverrideReq	FaultAlarm and OverrideOnFault	Description
true	either	either	Hand mode Hand is set to true Override is cleared to false
false	true	either	Override mode Hand is cleared to false Override is set to true
false	either	true	Override mode Hand is cleared to false Override is set to true

When Override is set, it takes precedence over Program and Operator control. The following table describes how the Override mode affects the commanded state.

Override	Override State	Description
true	2	Command0Status is cleared to false Command1Status is cleared to false Command2Status is set to true
true	1	Command0Status is cleared to false Command1Status is set to true Command2Status is cleared to false

Override	Override State	Description
true	0	Command0Status is set to true Command1Status is cleared to false Command2Status is cleared to false

If OverrideState is invalid, the instruction sets the appropriate bit in Status and does not enter the override state.

When Hand is true, it takes precedence over Program and Operator control. The following table describes how the Hand mode affects the commanded state.

Hand	HandFB0	HandFB1	HandFB2	Description
true	false	false	true	Command0Status is cleared to false Command1Status is cleared to false Command2Status is set to true
true	false	true	false	Command0Status is cleared to false Command1Status is set to true Command2Status is cleared to false
true	true	false	false	Command0Status is set to true Command1Status is cleared to false Command2Status is cleared to false

If more than one HandFB input is true, the instruction sets the appropriate bit in Status and, if Hand is true, the instruction holds the previous state.

Output State

The D3SD output state is based on the state of the command status.

CommandStatus	Output State
Command0Status is true	Out0 = Out0State0 Out1 = Out1State0 Out2 = Out2State0
Command0Status is true and FB0 = FB0State0 and FB1 = FB1State0 and FB2 = FB2State0 and FB3 = FB3State0	Stop and clear the fault timer. Device0State is set to true
Command1Status is true	Out0 = Out0State1

CommandStatus	Output State
	Out1 = Out1State1 Out2 = Out2State1
Command1Status is true and FB0 = FB0State1 and FB1 = FB1State1 and FB2 = FB2State1 and FB3 = FB3State1	Stop and clear the fault timer. Device1State is set to true
Command2Status is true	Out0 = Out0State2 Out1 = Out1State2 Out2 = Out2State2
Command2Status is true and FB0 = FB0State2 and FB1 = FB1State2 and FB2 = FB2State2 and FB3 = FB3State2	Stop and clear the fault timer. Device2State is set to true

Fault Alarm Conditions

The D3SD instruction checks for these fault alarm conditions.

Fault alarm condition resulting from	Rules
Device state was commanded to change, but the feedback did not indicate that the desired state was actually reached within the FaultTime	Start the fault timer when $Command0Status_n \neq Command0Status_{n-1}$ or $Command1Status_n \neq Command1Status_{n-1}$ or $Command2Status_n \neq Command2Status_{n-1}$ Set FaultAlarm when the fault timer done and FaultTime > 0.0
The device unexpectedly left a state (according to the feedback) without being commanded to	Set FaultAlarm to true when the fault timer is not timing and one of the following conditions is satisfied: Command0Status is true and Device0State is false Command1Status is true and Device1State is false Command2Status is true and Device2State is false

If there is no fault present, FaultAlarm is cleared to false if one of the following conditions is met:

- Command0Status is true and Device0State is true
- Command1Status is true and Device1State is true

- Command2Status is true and Device2State is true
- FaultTime ≤ 0

FaultAlarm cannot be cleared to false when FaultAlarmLatch is true, unless FaultAlmUnlatch is true and no fault is present.

Mode Alarm Conditions

The mode alarm reminds an operator that a device has been left in Operator control. The mode alarm only turns on when, in Operator control mode, the program tries to change the state of the device from the operator's commanded state. The alarm does not turn on if an operator places a device in Operator control mode and changes the state. The D3SD instruction checks for mode alarm conditions, using these rules.

ModeAlarm is	When
true	Prog2Command ≠ $Prog2Command_{n-1}$ **and** Prog2Command ≠ Command2Status **or** Prog1Command ≠ $Prog1Command_{n-1}$ **and** Prog1Command ≠ Command1Status **or** Prog0Command ≠ $Prog1Command_{n-1}$ **and** Prog0Command ≠ Command0Status
false	Prog2Command = Command2Status and Prog1Command = Command1Status and Prog0Command = Command0Status or The device is in Override, Hand, or Program control mode

Discrete 2-State Device (D2SD)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, Compact GuardLogix 5380, CompactLogix 5380, ControlLogix 5580, GuardLogix 5580, and ControlLogix 5590 controllers.

The Discrete 2-State Device (D2SD) instruction controls a discrete device which has only two possible states (such as on/off or open/closed).

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram logic.

Function Block

Structured Text

D2SD(D2SD_tag)

Operands

There are data conversion rules for mixed data types within an instruction. See Data Conversions on page .

Structured Text

Operand	Type	Format	Description
D2SD tag	DISCRETE_2STATE	Structure	D2SD structure

See *Structured Text Syntax* for more information on the syntax of expressions within structured text.

Function Block

Operand	Type	Format	Description
D2SD tag	DISCRETE_2STATE	Structure	D2SD structure

DISCRETE_2STATE Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If false, the instruction does not execute and outputs are not updated. Default is true.

Input Parameter	Data Type	Description
ProgCommand	BOOL	Used to determine CommandStatus when the device is in Program control. When true, the device is commanded to the 1 state; when false, the device is commanded to the 0 state. Default is false.
Oper0Req	BOOL	Operator state 0 request. Set by the operator interface to place the device in the 0 state when the device is in Operator control. Default is false.
Oper1Req	BOOL	Operator state 1 request. Set by the operator interface to place the device in the 1 state when the device is in Operator control. Default is false.
State0Perm	BOOL	State 0 permissive. Unless in Hand or Override mode, this input must be set for the device to enter the 0 state. This input has no effect for a device already in the 0 state. Default is true.
State1Perm	BOOL	State 1 permissive. Unless in the Hand or Override mode, this input must be set for the device to enter the 1 state. This input has no effect for a device already in the 1 state. Default is true.
FB0	BOOL	The first feedback input available to the D2SD instruction. Default is false.
FB1	BOOL	The second feedback input available to the D2SD instruction. Default is false.
HandFB	BOOL	Hand feedback input. This input is from a field hand/off/auto station and it shows the requested state of the field device. When true, the field device is being requested to enter the 1 state; when false, the field device is being requested to enter the 0 state. Default is false.
FaultTime	REAL	Fault time value. Configure the value in seconds of the time to allow the

Input Parameter	Data Type	Description
		device to reach a newly commanded state. Set FaultTime = 0 to disable the fault timer. If this value is invalid, the instruction assumes a value of zero and sets the appropriate bit in Status. Valid = any float ≥ 0.0 Default = 0.0
FaultAlarmLatch	BOOL	Fault alarm latch input. When true and FaultAlarm is true, latch FaultAlarm. To unlatch FaultAlarm set FaultAlmUnlatch to true or clear FaultAlarmLatch to false. Default is false.
FaultAlmUnLatch	BOOL	Fault alarm unlatch input. Set FaultAlmUnLatch when FaultAlarmLatch is set to unlatch FaultAlarm. The instruction clears this input to false. Default is false.
OverrideOnInit	BOOL	Override on initialization request. If this bit is true, then during instruction first scan, the 2-state device is placed in Operator control, Override is set to true, and Hand is cleared to false. If ProgHandReq is true, then Override is cleared to false and Hand is set to true. Default is false.
OverrideOnFault	BOOL	Override on fault request. Set OverrideOnFault to true if the device should go to Override mode and enter the OverrideState on a fault alarm. After the fault alarm is removed, the 2-state device is placed in Operator control. Default is false.
OutReverse	BOOL	Reverse default out state. The default state of Out is cleared to false when commanded to state 0, and set to true when commanded to state 1. When OutReverse is true, Out is set to true when commanded to state 0, and cleared to false when commanded to state 1. Default is false.

Input Parameter	Data Type	Description
OverrideState	BOOL	Override state input. Configure this value to specify the state of the device when the device is in Override mode. True indicates the device should go to the 1 state; false indicates the device should go to the 0 state. Default is false.
FB0State0	BOOL	Feedback 0 state 0 input. Configure the state of the FB0 when the device is in the 0 state. Default is false.
FB0State1	BOOL	Feedback 0 state 1 input. Configure the state of the FB0 when the device is in the 1 state. Default is false.
FB1State0	BOOL	Feedback 1 state 0 input. Configure the state of the FB1 when the device is in the 0 state. Default is false.
FB1State1	BOOL	Feedback 1 state 1 input. Configure the state of the FB1 when the device is in the 1 state. Default is false.
ProgProgReq	BOOL	Program program request. Set to true by the user program to request Program control. Ignored if ProgOperReq is true. Holding this true and ProgOperReq false locks the instruction into Program control. Default is false.
ProgOperReq	BOOL	Program operator request. Set to true by the user program to request Operator control. Holding this true locks the instruction into Operator control. Default is false.
ProgOverrideReq	BOOL	Program override request. Set to true by the user program to request the device to enter Override mode. Ignored if ProgHandReq is true. Default is false.
ProgHandReq	BOOL	Program hand request. Set to true by the user program to request the device to enter Hand mode. Default is false.

Input Parameter	Data Type	Description
OperProgReq	BOOL	Operator program request. Set to true by the operator interface to request Program control. The instruction clears this input to false. Default is false.
OperOperReq	BOOL	Operator operator request. Set to true by the operator interface to request Operator control. The instruction clears this input to false. Default is false.
ProgValueReset	BOOL	Reset program control values. When true, all the program request inputs are cleared to false at each execution of the instruction. Default is false.

Output Parameter	Data Type	Description
EnableOut	BOOL	Indicates if instruction is enabled.
Out	BOOL	The output of the 2-state instruction.
Device0State	BOOL	Device 0 state output. Set to true when the device is commanded to the 0 state and the feedbacks indicate the device really is in the 0 state.
Device1State	BOOL	Device 1 state output. Set to true when the device is commanded to the 1 state and the feedbacks indicate the device really is in the 1 state.
CommandStatus	BOOL	Command status output. Set to true when the device is being commanded to the 1 state and cleared when the device is being commanded to the 0 state.
FaultAlarm	BOOL	Fault alarm output. Set to true if the device was commanded to a new state and the FaultTime has expired without the feedbacks indicating that the new state has actually been reached. Also set to true if, after reaching a commanded state, the feedbacks suddenly indicate

Output Parameter	Data Type	Description
		that the device is no longer in the commanded state.
ModeAlarm	BOOL	Mode alarm output. Set to true if the device is in Operator control and a program command changes to a state which is different from the state currently commanded by the operator. This alarm is intended as a reminder that a device was left in Operator control.
ProgOper	BOOL	Program/Operator control indicator. True when in Program control. False when in Operator control.
Override	BOOL	Override mode. True when the device is in the Override mode.
Hand	BOOL	Hand mode. True when the device is in the Hand mode.
Status	DINT	Status of the function block.
InstructFault (Status.0)	BOOL	The instruction detected one of the following execution errors. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.
FaultTimeInv (Status.1)	BOOL	Invalid FaultTime value. The instruction sets FaultTime = 0.
OperReqInv (Status.2)	BOOL	Both operator state request bits are true.

Description

The D2SD instruction controls a discrete device which has only two possible states (such as on/off or open/closed). Typical discrete devices of this nature include motors, pumps, and solenoid valves.

Monitoring the D2SD Instruction

There is an operator faceplate available for the D2SD instruction.

Affects Math Status Flags

No

Major/Minor Faults

None specific to this instruction. See Common Attributes on page for operand-related faults.

Execution

Function Block

Condition/State	Action Taken
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is true	EnableIn and EnableOut bits are set to true. The instruction executes.
Instruction first run	Set ProgOper to Operator Mode. Set CommandStatus to False.
Instruction first scan	Set EnableOut to true. ModeAlarm and operator request inputs are cleared to false, If ProgValueReset is true, all the program request inputs are cleared to false. When OverrideOnInit is true, ProgOper is cleared to false (Operator control). If ProgHandReq is cleared and OverrideOnInit is set, clear Hand and set Override (Override mode). If ProgHandReq is set, set Hand and clear Override (Hand mode).
Postscan	EnableIn and EnableOut bits are cleared to false.

Structured Text

In Structured Text, EnableIn is always true during normal scan. Therefore, if the instruction is in the control path activated by the logic it will execute.

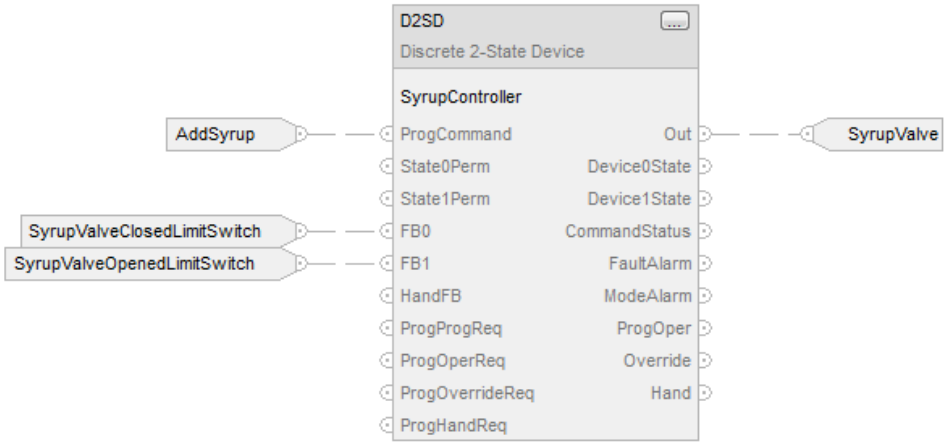
Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.
Normal Execution	See Tag.EnableIn is true in the Function Block table.
Postscan	See Postscan in the Function Block table.

Examples

SD instruction is typically used to control on-off or open-close devices such as pumps or solenoid valves. In this example, the D2SD instruction controls a solenoid valve adding corn

syrup to a batch tank. As long as the D2SD instruction is in Program control, the valve opens when the AddSyrup input is set. The operator can also take Operator control of the valve to open or close it if necessary The solenoid valve in this example has limit switches that indicate when the valve is fully closed or opened. These switches are wired into the FB0 and FB1 feedback inputs. This allows the D2SD instruction to generate a FaultAlarm if the solenoid valve does not reach the commanded state within the configured FaultTime.

Function Block

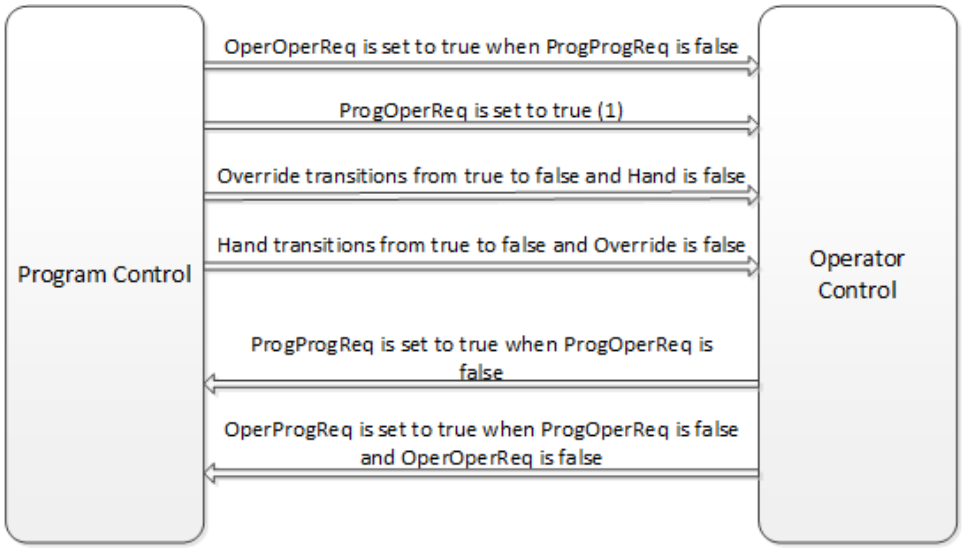


Structured Text

```
SyrupController.ProgCommand := AddSyrup;
SyrupController.FB0 := SyrupValveClosedLimitSwitch;
SyrupController.FB1 := SyrupValveOpenedLimitSwitch;
D2SD(SyrupController);
SyrupValve := SyrupController.Out;
```

Switch Between Program Control and Operator Control

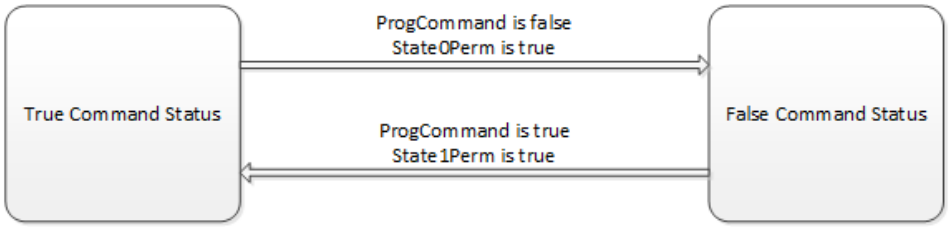
The following diagram shows how the D2SD instruction changes between Program control and Operator control.



(1) The instruction remains in Operator control mode when ProgOperReq is true.

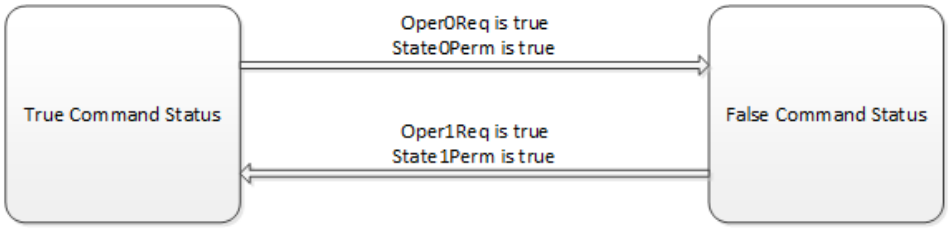
Commanded State in Program Control

The following diagram shows how the D2SD instruction operates when in Program control.



Commanded State in Operator Control

The following diagram shows how the D2SD instruction operates when in Operator control.



If both OperOReq and Oper1Req are true:

- The instruction sets the appropriate bit in Status to true
- If Override and Hand are false, the instruction holds the previous state.

After every instruction execution, the instruction:

- Clears all the operator request inputs to false
- If ProgValueReset is true, clears all the program request inputs to false

Hand Mode or Override Mode

The following table describes how the D2SD instruction determines whether to operate in Hand or Override mode.

ProgHandReq	ProgOverrideReq	FaultAlarm and OverrideOnFault	Description
true	either	either	Hand mode Hand is set to true Override is cleared to false
false	true	either	Override mode Hand is cleared to false Override is set to true
false	either	true	Override mode Hand is cleared to false Override is set to true

When the instruction is in Override mode, CommandStatus = OverrideState.

When the instruction is in Hand mode, CommandStatus = HandFB.

Output State

The D2SD output state is based on the state of the command status.

CommandStatus	Output State
false	If OutReverse is false, Out is cleared to false If OutReverse is true, Out is set to true
true	If OutReverse is false, Out is set to rue If OutReverse is true, Out is cleared to false
false and FB0 = FB0State0 and FB1 = FB1State0	The fault timer is stopped and cleared to 0 Device0State is set to true
true and FB0 = FB0State1 and FB1 = FB1State1	The fault timer is stopped and cleared to 0 Device1State is set to true

Fault Alarm Conditions

The D2SD instruction checks for these fault alarm conditions.

Fault alarm condition resulting from	Rules
Device state was commanded to change, but the feedback did not indicate that the desired state was actually reached within the FaultTime	Start the fault timer when CommandStatus $_n \neq$ CommandStatus$_{n-1}$ Set FaultAlarm when faulttimer is done and FaultTime > 0.0
The device unexpectedly left a state (according to the feedback) without being commanded to	Set FaultAlarm to true when the fault timer is not timing and one of the following conditions is satisfied: CommandStatus is false and Device0State is false CommandStatus is true and Device1State is false

FaultAlarm is cleared to false if one of the following conditions is met:

- CommandStatus is false and Device0State is true
- CommandStatus is true and Device1State is true
- FaultTime $\leq$ 0

FaultAlarm cannot be cleared to false when FaultAlarmLatch is true, unless FaultAlmUnlatch is true and no fault is present.

Mode Alarm Conditions

The mode alarm reminds an operator that a device has been left in Operator control. The mode alarm only turns on when, in Operator control mode, the program tries to change the state of the device from the operator's commanded state. The alarm does not turn on if an operator places a device in Operator control mode and changes the state. The D2SD instruction checks for mode alarm conditions, using these rules.

ModeAlarm	When
True	ProgCommand $_n$ ≠ ProgCommandn-1 and ProgCommand $_n$ ≠ CommandStatus
False	ProgCommand = CommandStatus or the device is in Override, Hand, or Program control mode

Deadtime (DEDT)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, Compact GuardLogix 5380, CompactLogix 5380, ControlLogix 5580, GuardLogix 5580, and ControlLogix 5590 controllers.

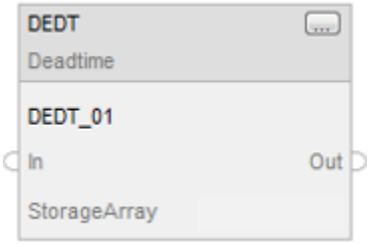
The Deadtime (DEDT) instruction performs a delay of a single input. You select the amount of deadtime delay.

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram logic.

Function Block



Structured Text

DEDT(DEDT_tag,storage);

Operands

Structured Text

Operand	Type	Format	Description
DEDT tag	DEADTIME	structure	DEDT structure

storage	REAL	array	deadtime buffer

See *Structured Text Syntax* for more information on the syntax of expressions within structured text.

Function Block

Operand	Type	Format	Description
DEDT tag	DEADTIME	structure	DEDT structure
storage	REAL	array	deadtime buffer

DEADTIME Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If false, the instruction does not execute and outputs are not updated. Default is true.
In	REAL	The analog signal input to the instruction. Valid = any float Default = 0.0
InFault	BOOL	Bad health indicator for the input. If the input value is read from an analog input, then InFault is controlled by fault status on the analog input. If true, InFault indicates the input signal has an error, the instruction sets the appropriate bit in Status, the control algorithm is not executed, and Out is held. Default is false. false = good health
Deadtime	REAL	Deadtime input to the instruction. Enter the deadtime in seconds. If this value is invalid, the instruction assumes a value of zero and sets the appropriate bit in Status. Valid = 0.0 to (StorageArray size * DeltaT) Default = 0.0
Gain	REAL	Gain input to the instruction. The value of In is multiplied by this value. This allows simulation of a process gain.

Input Parameter	Data Type	Description
		Valid = any float Default = 1.0
Bias	REAL	Bias input to the instruction. The value of In multiplied by the Gain is added to this value. This allows simulation of an ambient condition. Valid = any float Default = 0.0
TimingMode	DINT	Selects timing execution mode. 0 = Period mode 1 = oversample mode 2 = real time sampling mode Valid = 0 to 2 Default = 0 For more information about timing modes, see Function Block Attributes.
OversampleDT	REAL	Execution time for oversample mode. Valid = 0 to 4194.303 seconds Default = 0
RTSTime	DINT	Module update period for real time sampling mode Valid = 1 to 32,767ms Default = 1
RTSTimeStamp	DINT	Module time stamp value for real time sampling mode. Valid = 0 to 32,767ms Default = 0

Output Parameter	Data Type	Description
EnableOut	BOOL	Indicates if instruction is enabled. Cleared to false if Out overflows.
Out	REAL	The calculated output of the deadtime algorithm.
DeltaT	REAL	Elapsed time between updates. This is the elapsed time in seconds used by the control algorithm to calculate the process output.
Status	DINT	Status of the function block.

Output Parameter	Data Type	Description
InstructFault (Status.0)	BOOL	The instruction detected one of the following execution errors. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.
InFaulted (Status.1)	BOOL	In health is bad.
DeadtimeInv (Status.2)	BOOL	Invalid Deadtime value.
TimingMode (Status.27)	BOOL	Invalid TimingMode value. For more information about timing modes, see Function Block Attributes.
RTSMissed (Status.28)	BOOL	Only used in real time sampling mode. Set when ABS(DeltaT - RTSTime) > 1 millisecond.
RTSTimeInv (Status.29)	BOOL	Invalid RTSTime value.
RTSTimeStampInv (Status.30)	BOOL	Invalid RTSTimeStamp value.
DeltaTInv (Status.31)	BOOL	Invalid DeltaT value.

Description

The DEDT instruction uses a data buffer to store delayed data, thereby allowing any length deadtime desired. The DEDT instruction is designed to execute in a task where the scan rate remains constant.

To use the DEDT instruction, create a storage array to store the deadtime buffer to hold the samples of (In x Gain) + Bias. The storage array should be large enough to hold the largest desired deadtime, using this formula:

StorageArray Size Needed = Maximum Deadtime (secs) / DeltaT (secs)

Servicing the Deadtime Buffer

During runtime, the instruction checks for a valid Deadtime. Deadtime must be between 0.0 and (StorageArray Size x DeltaT).

If the Deadtime is invalid, the instruction sets an appropriate Status bit and sets Out = (In x Gain) + Bias.

The deadtime buffer functions as a first-in, first-out buffer. Every time the deadtime algorithm executes, the oldest value in the deadtime buffer is moved into Out. The remaining values in the buffer shift downward and the value ((In x Gain) + Bias) is moved to the beginning of the deadtime buffer. A new value that is placed in the deadtime buffer appears in the Out after Deadtime seconds.

The number of array elements required to perform the programmed delay is calculated by dividing Deadtime by DeltaT. If Deadtime is not evenly divisible by DeltaT, then the number of array elements and the programmed delay are rounded to the nearest increment of DeltaT. For

example, to find the number of array elements required to perform the programmed delay given Deadtime = 4.25s and DeltaT = 0.50s:

4.25s / 0.50s = 8.5
rounds up to 9 array elements required

The actual delay applied to the input in this example is:

number of array elements x DeltaT = programmed delay or
9 x 0.5s = 4.5s

Runtime changes to either Deadtime or DeltaT change the point in which values are moved out of the buffer. The number of elements required to perform the programmed delay can either increase or decrease. Prior to servicing the deadtime buffer, the following updates occur:

If the number of required elements needs to increase, the new buffer elements are populated with the oldest value in the current deadtime buffer.

If the number of required elements needs to decrease, the oldest elements of the current deadtime buffer are discarded.

Instruction Behavior on InFault Transition

When InFault is true (bad), the instruction suspends execution, holds the last output, and sets the appropriate bit in Status.

When InFault transitions from true to false, the instruction sets all values in the deadtime buffer equal to In x Gain + Bias.

Affects Math Status Flags

No

Major/Minor Faults

None specific to this instruction. See Common Attributes on page for operand-related faults.

Execution

Function Block

Condition/State	Action Taken
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is true	EnableIn and EnableOut bits are set to true. The instruction executes.
Instruction first run	N/A
Instruction first scan	N/A The instruction does not execute, but does validate input parameters.

Condition/State	Action Taken
Postscan	EnableIn and EnableOut bits are cleared to false.

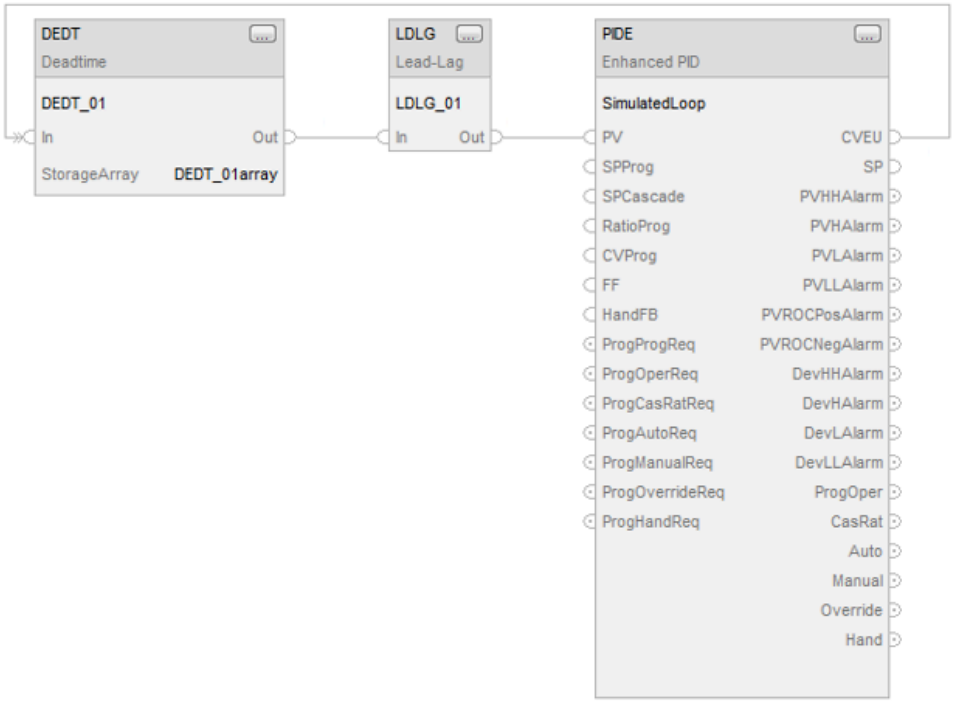
Structured Text

Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.
Normal Execution	See Tag.EnableIn is true in the Function Block table.
Postscan	See Postscan in the Function Block table.

Example

In this example, the DEDT instruction simulates a deadtime delay in a simulated process. The output of the PIDE instruction is passed through a deadtime delay and a first-order lag to simulate the process. The array DEDT_01array is a REAL array with 100 elements to support a deadtime of up to 100 samples. For example, if this routine executes every 100 msec, the array would support a deadtime of up to 10 seconds.

Function Block



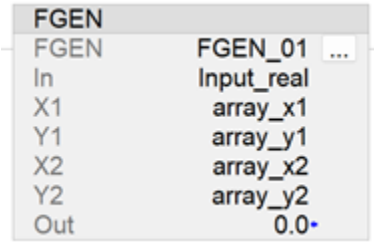
Function Generator (FGEN)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, Compact GuardLogix 5380, CompactLogix 5380, ControlLogix 5580, GuardLogix 5580, and ControlLogix 5590 controllers.

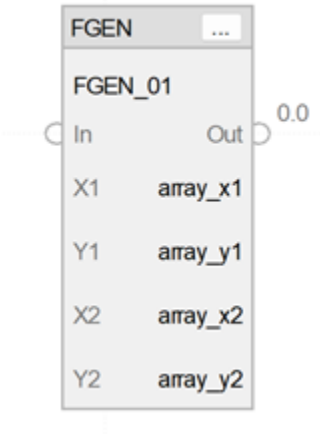
The Function Generator (FGEN) instruction converts an input based on a piece-wise linear function.

Available Languages

Ladder Diagram



Function Block



Structured Text

FGEN(FGEN_tag,X1,Y1,X2,Y2);

Operands

Ladder Diagram

Operand	Type	Format	Description
FGEN tag	FUNCTION_ GENERATOR	structure	FGEN structure
In	REAL	tag / immediate	The analog signal input to the instruction. Valid = any float
X1	REAL	array	X-axis array, table one. Combine with the Y-axis array, table one to define

Operand	Type	Format	Description
			the points of the first piece-wise linear curve. Valid = any float
Y1	REAL	array	Y-axis array, table one. Combine with the X-axis array, table one to define the points of the first piece-wise linear curve. Valid = any float
X2	REAL	array	(optional) X-axis array, table two. Combine with the Y-axis array, table two to define the points of the second piece-wise linear curve. Valid = any float
Y2	REAL	array	(optional) Y-axis array, table two. Combine with the X-axis array, table two to define the points of the second piece-wise linear curve. Valid = any float

Function Block

Operand	Type	Format	Description
FGEN tag	FUNCTION_ GENERATOR	structure	FGEN structure
X1	REAL	array	X-axis array, table one. Combine with the Y-axis array, table one to define the points of the first piece-wise linear curve. Valid = any float
Y1	REAL	array	Y-axis array, table one. Combine with the X-axis array, table one to define the points of the first piece-wise linear curve. Valid = any float
X2	REAL	array	(optional) X-axis array, table two. Combine with the Y-axis array, table two to define

Operand	Type	Format	Description
			the points of the second piece-wise linear curve. Valid = any float
Y2	REAL	array	(optional) Y-axis array, table two. Combine with the X-axis array, table two to define the points of the second piece-wise linear curve. Valid = any float

Structured Text

Operand	Type	Format	Description
FGEN tag	FUNCTION_ GENERATOR	structure	FGEN structure
X1	REAL	array	X-axis array, table one. Combine with the Y-axis array, table one to define the points of the first piece-wise linear curve. Valid = any float
Y1	REAL	array	Y-axis array, table one. Combine with the X-axis array, table one to define the points of the first piece-wise linear curve. Valid = any float
X2	REAL	array	(optional) X-axis array, table two. Combine with the Y-axis array, table two to define the points of the second piece-wise linear curve. Valid = any float
Y2	REAL	array	(optional) Y-axis array, table two. Combine with the X-axis array, table two to define the points of the second piece-wise linear curve. Valid = any float

See *Structured Text Syntax* for more information on the syntax of expressions within structured text.

FUNCTION_GENERATOR Structure

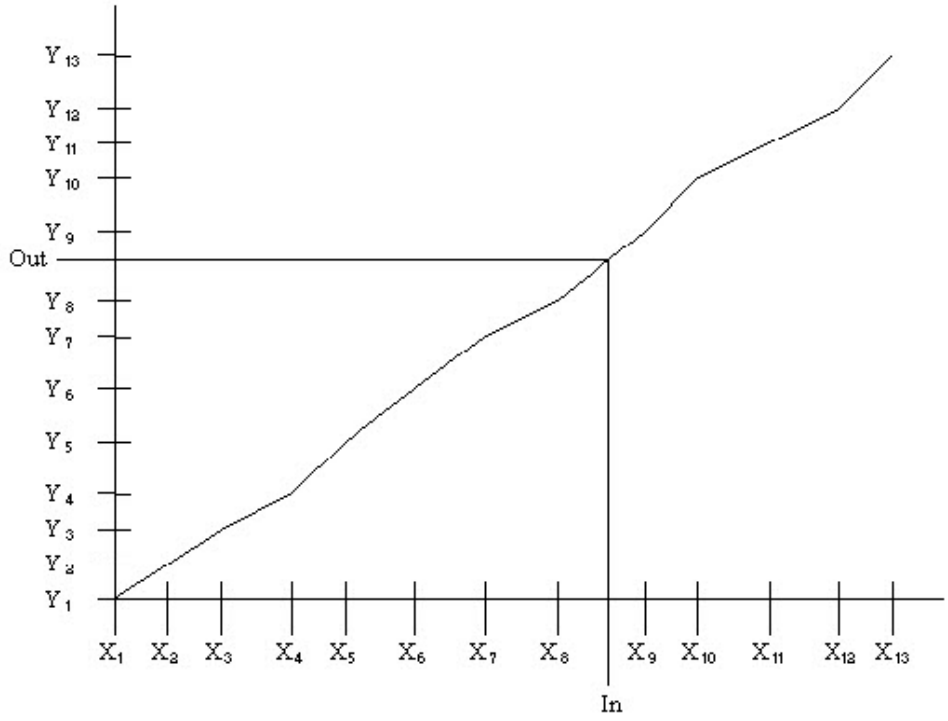
Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If false, the instruction does not execute and outputs are not updated. Default is true.
In	REAL	The analog signal input to the instruction. Valid = any float Default = 0.0
XY1Size	DINT	Number of points in the piece-wise linear curve to use from table one. If the value is less than one and Select is cleared, the instruction sets the appropriate bit in Status and the output is not changed. Valid = 1 to (smallest of X1 and Y1 array sizes) Default = 1
XY2Size	DINT	Number of points in the piece-wise linear curve to use from table two. If the value is less than one and Select is set, the instruction sets the appropriate bit in Status and the output is not changed. Valid = 0 to (smallest of X2 and Y2 array sizes) Default = 0
Select	BOOL	This input determines which table to use. When cleared, the instruction uses table one; when set, the instruction uses table two. Default is cleared.

Output Parameter	Data Type	Description
EnableOut	BOOL	Indicates if instruction is enabled. Cleared to false on overflow
Out	REAL	Output of the instruction.
Status	DINT	Status of the function block.
InstructFault (Status.0)	BOOL	Instruction generated a fault.
XY1SizeInv (Status.1)	BOOL	Size of table 1 is invalid or not compatible with the array size.

Output Parameter	Data Type	Description
XY2SizeInv (Status.2)	BOOL	Size of table 2 is invalid or not compatible with the array size.
XisOutofOrder (Status.3)	BOOL	The X parameters are not sorted.

Description

The following illustration shows how the FGEN instruction converts a twelve-segment curve.



The X-axis parameters must follow the relationship:

X[1] < X[2] < X[3] < ... < X[XY<n>Size],

where XY<n>Size > 1 and is a number of points in the piece-wise linear curve and where n is 1 or 2 for the table selected. You must create sorted X-axis elements in the X arrays.

The Select input determines which table to use for the instruction. When the instruction is executing on one table, you can modify the values in the other table. Change the state of Select to execute with the other table.

Before calculating Out, the X axis parameters are scanned. If they are not sorted in ascending order, the appropriate bit in Status is set and Out remains unchanged. Also, if XY1Size or XY2Size is invalid, the instruction sets the appropriate bit in Status and leaves Out unchanged.

The instruction uses this algorithm to calculate Out based on In:

- When In ≤ X[1], set Out = Y[1]
- When In > X[XY<n>Size], set Out = Y[XY<n>Size]
- When X[n] < In ≤ X[n+1], calculate Out = ((Y[n+1]-Yn)/ (X[n+1]-Xn))*(In-Xn)+Yn

Affects Math Status Flags

No

Major/Minor Fault

None specific to this instruction. See Common Attributes on page for operand-related faults.

Execution

Ladder Diagram

Condition	Function Block Action
Prescan	Rung-condition-out is cleared to false.
Rung-condition-in is false	Set Rung-condition-out to Rung-condition-in.
Rung-condition-in is true	Set Rung-condition-out to Rung-condition-in. Execute FGEN algorithm.
Postscan	Rung-condition-out is cleared to false.

Function Block Diagram

Condition	Function Block Action
Prescan	EnableIn and EnableOut bits are set to false.
EnableIn is false	EnableIn and EnableOut bits are set to false.
EnableIn is true	EnableIn and EnableOut bits are set to true. Execute FGEN algorithm.
Instruction first run	N/A
Instruction first scan	Set the internal Out to 0.0 Execute FGEN algorithm.
Postscan	EnableIn and EnableOut bits are set to false.

Structured Text

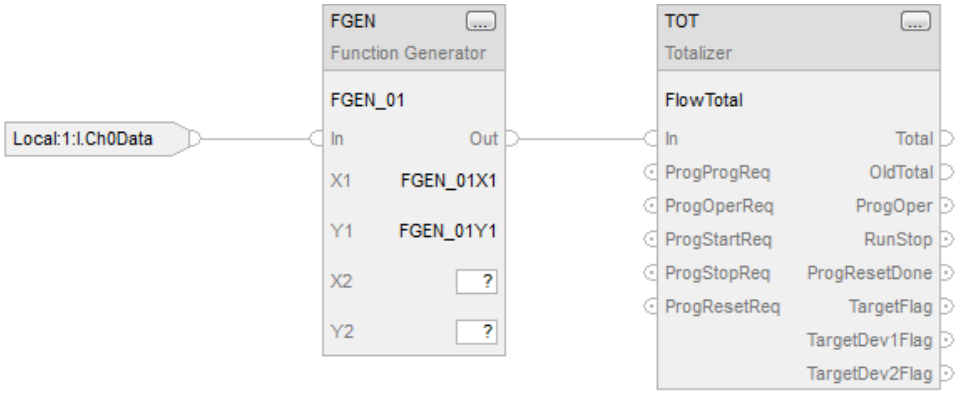
Condition/State	Action Taken
Prescan	EnableIn and EnableOut bits are set to false.
Normal Execution	EnableIn and EnableOut bits are set to true. Execute FGEN algorithm.
Postscan	EnableIn and EnableOut bits are set to false.

Examples

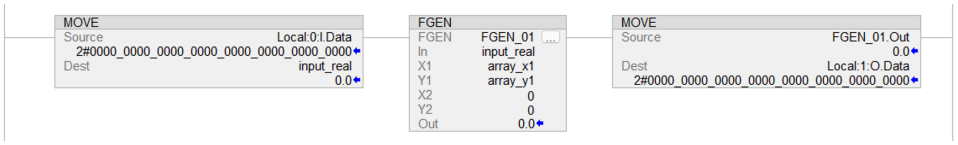
Example 1

In this example, the FGEN instruction characterizes a flow signal which is then totalized using a TOT instruction. The FGEN_01X1 and FGEN_01Y1 arrays are REAL arrays of 10 elements each to support up to a 9 segment curve. You can use arrays of any size to support a curve of any desired number of segments.

Function Block



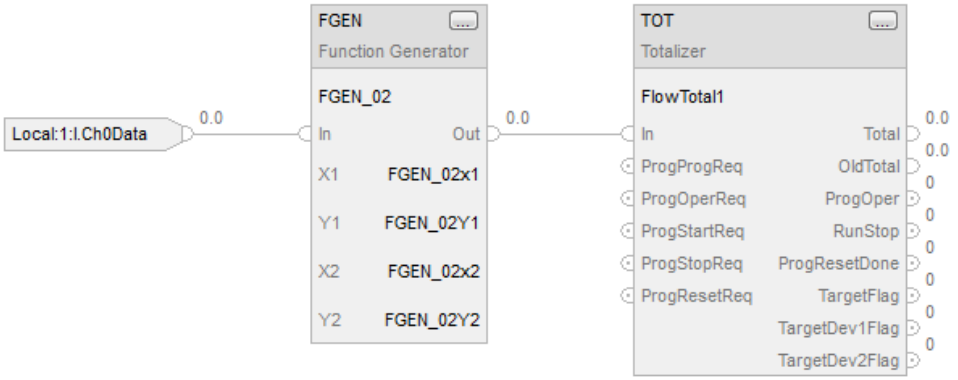
Ladder Diagram



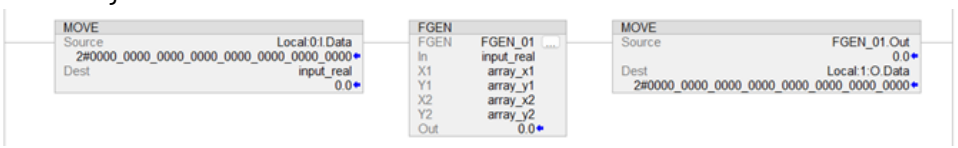
Example 2

This example passes optional parameters to FGEN instruction.

Function Block



Ladder Diagram



Lead-Lag (LDLG)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, Compact GuardLogix 5380, CompactLogix 5380, ControlLogix 5580, GuardLogix 5580, and ControlLogix 5590 controllers.

The Lead-Lag (LDLG) instruction provides a phase lead-lag compensation for an input signal. This instruction is typically used for feedforward PID control or for process simulations.

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram.

Function Block

Structured Text

LDLG(LDLG_tag);

Operands

Function Block

Operand	Type	Format	Description
LDLG tag	LEAD_LAG	Structure	LDLG structure

Structured Text

Operand	Type	Format	Description
LDLG tag	LEAD_LAG	Structure	LDLG structure

See *Structured Text Syntax* for more information on the syntax of expressions within structured text.

LEAD_LAG Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated.

Input Parameter	Data Type	Description
		Default is set.
In	REAL	The analog signal input to the instruction. Valid = any float Default = 0.0
Initialize	BOOL	Request to initialize filter control algorithm. When Initialize is set, Out = (In x Gain) + Bias. Default = cleared
Lead	REAL	The lead time in seconds. Set Lead = 0.0 to disable the lead control algorithm. If Lead < 0.0, the instruction sets the appropriate bit in Status and limits Lead to 0.0. If Lead > maximum positive float, the instruction sets the appropriate bit in Status. Valid = any float ≥ 0.0 Default = 0.0
Lag	REAL	The lag time in seconds. The minimum lag time is DeltaT/2. If Lag < DeltaT/2, the instruction sets the appropriate bit in Status and limits Lag to DeltaT/2. If Lag > maximum positive float, the instruction sets the appropriate bit in Status. Valid = any float ≥ DeltaT/2 Default = 0.0
Gain	REAL	The process gain multiplier. This value allows the simulation of a process gain. The In signal is multiplied by this value. I = (In x Gain) + Bias Valid = any float Default = 1.0
Bias	REAL	The process offset level. This value allows the simulation of an ambient condition. This value is summed with the results of the multiplication of In times Gain. I = (In x Gain) + Bias Valid = any float Default = 0.0
TimingMode	DINT	Selects timing execution mode. 0 = Periodic rate 1 = Oversample mode

Input Parameter	Data Type	Description
		2 = Real-time sampling mode Valid = 0 to 2 Default = 0 For more information about timing modes, see Function Block Attributes.
OversampleDT	REAL	Execution time for oversample mode. Valid = 0 to 4194.303 seconds Default = 0
RTSTime	DINT	Module update period for real time sampling mode Valid = 1 to 32,767ms Default = 1
RTSTimeStamp	DINT	Module time stamp value for real time sampling mode. Valid = 0 to 32,767ms Default = 0

Output Parameter	Data Type	Description
EnableOut	BOOL	Enable output.
Out	REAL	The calculated output of the algorithm. Math status flags are used for this output.
DeltaT	REAL	Elapsed time between updates. This is the elapsed time in seconds used by the control algorithm to calculate the process output.
Status	DINT	Status of the function block.
InstructFault (Status.0)	BOOL	The instruction detected one of the following execution errors. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.
LeadInv (Status.1)	BOOL	Lead < minimum value or Lead > maximum value.
LagInv (Status.2)	BOOL	Lag < minimum value or Lag > maximum value.
TimingModeInv (Status.27)	BOOL	Invalid TimingMode value.

Output Parameter	Data Type	Description
		For more information about timing modes, see Function Block Attributes.
RTSMissed (Status.28)	BOOL	Only used in real time sampling mode. Set when ABS | DeltaT - RTSTime | > 1 (.001 second).
RTSTimeInv (Status.29)	BOOL	Invalid RTSTime value.
RTSTimeStampInv (Status.30)	BOOL	Invalid RTSTimeStamp value.
DeltaTInv (Status.31)	BOOL	Invalid DeltaT value.

Description

The LDLG instruction supports one lead and lag in series. The instruction also allows configurable gain and bias factors. The LDLG instruction is designed to execute in a task where the scan rate remains constant.

The LDLG instruction uses this equation:

$$H(s) = \left[\frac{1 + Lead \times s}{1 + Lag \times s}\right]$$

with these parameter limits:

Parameter	Limitations
Lead	LowLimit = 0.0 HighLimit = maximum positive float
Lag	LowLimit = DeltaT/2 (DeltaT is in seconds) HighLimit = maximum positive float

Whenever the value computed for the output is invalid, NAN, or ± INF, the instruction sets Out = the invalid value and sets the Math overflow status flag. When the value computed for the output becomes valid, the instruction initializes the internal parameters and sets Out = (In x Gain) + Bias.

Math Status Flags

No

Major/Minor Faults

None specific to this instruction. See Common Attributes on page for operand-related faults.

Execution

Function Block

Condition	Function Block Action
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is true	EnableIn and EnableOut bits are set to true. The instruction executes.
Instruction first run	N/A
Instruction first scan	Execute "Out = (In * Gain) + Bias". Recalculate Lead/Lag coefficients.
Postscan	EnableIn and EnableOut bits are cleared to false.

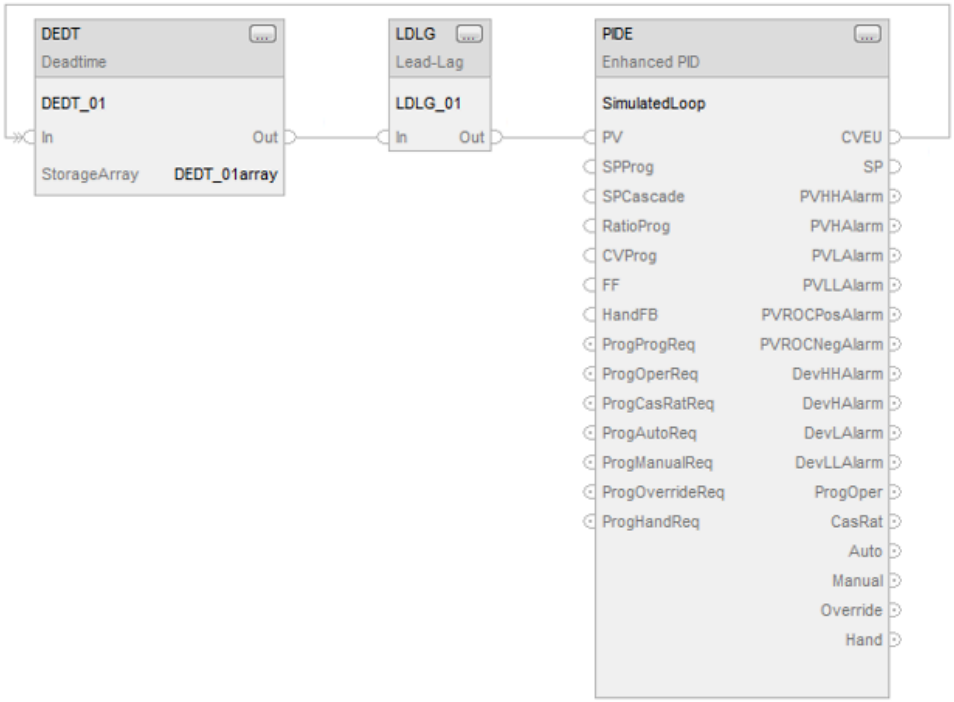
Structured Text

Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.
Normal Execution	See Tag.EnableIn is true in the Function Block table.
Postscan	See Postscan in the Function Block table.

Example

The LDLG instruction in this example adds a first-order lag to a simulated process. Optionally, you could enter a Gain on the LDLG instruction to simulate a process gain, and you could enter a Bias to simulate an ambient condition.

Function Block



Structured Text

```
DEDT_01.In := SimulatedLoop.CVEU;
DEDT(DEDT_01,DEDT_01_array);
LDLG_01.In := DEDT_01.Out;
LDLG(LDLG_01);
SimulatedLoop.PV := LDLG_01.Out;
PIDE(SimulatedLoop);
```

Enhanced PID (PIDE)

This information applies to the CompactLogix 5370, ControlLogix 5570, CompactLogix 5380, and ControlLogix 5580 controllers.

NOTE: The PIDE instruction is not supported in safety applications and is not supported on ControlLogix 5590 controllers. For ControlLogix 5590 controller projects, substitute the PPID instruction. See *Replacement Guidelines: Logix 5000 Controllers* (publication 1756-RM100) for information on migrating projects for use with ControlLogix 5590 controllers.

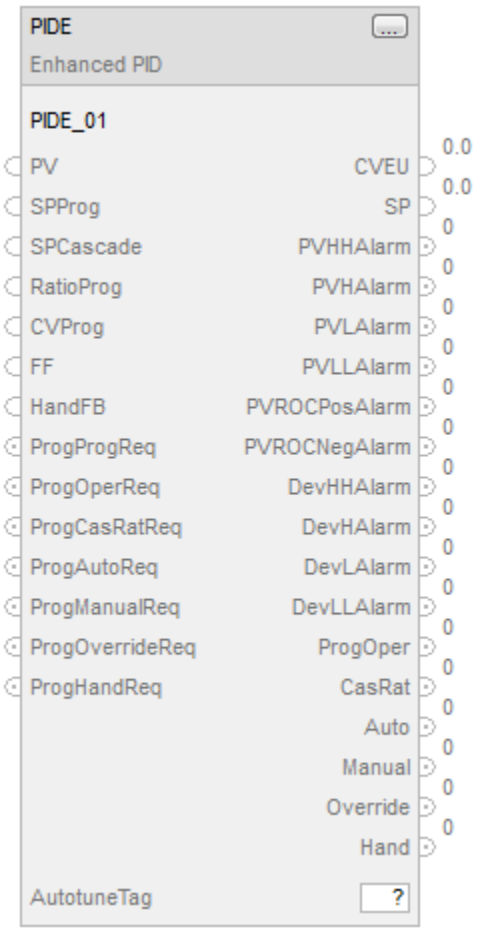
The Enhanced PID (PIDE) instruction provides enhanced capabilities over the standard PID instruction. The instruction uses the velocity form of the PID algorithm. The gain terms are applied to the change in the value of error or PV, not the value of error or PV.

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram.

Function Block



Structured Text

PIDE(PIDE_tag);

Operands

Function Block

Operand	Type	Format	Description
PIDE tag	PID_ENHANCED	structure	PIDE structure
autotune tag	PIDE_AUTOTUNE	structure	(optional) autotune structure

Structured Text

Operand	Type	Format	Description
PIDE tag	PID_ENHANCED	structure	PIDE structure

See *Structured Text Syntax* for more information on the syntax of expressions within structured text.

PIDE Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If false, the instruction does not execute and outputs are not updated. Default is true.
PV	REAL	Scaled process variable input. This value is typically read from an analog input module. Valid = any float Default = 0.0
PVFault	BOOL	PV bad health indicator. If PV is read from an analog input, then PVFault is normally controlled by the analog input fault status. When PVFault is true, it indicates the input signal has an error. Default is false = "good health"
PVEUMax	REAL	Maximum scaled value for PV. The value of PV and SP which corresponds to 100 percent span of the Process Variable. Valid = PVEUMin < PVEUMax ≤ maximum positive float Default = 100.0
PVEUMin	REAL	Minimum scaled value for PV. The value of PV and SP which corresponds to 0 percent span of the Process Variable. Valid = maximum negative float ≤ PVEUMin < PVEUMax Default = 0.0
SPProg	REAL	SP program value, scaled in PV units. SP is set to this value when in Program control and not Cascade/Ratio mode. If the value of SPProg < SPLLimit or > SPHLimit, the instruction sets the appropriate bit in Status and limits the value used for SP. Valid = SPLLimit to SPHLimit Default = 0.0

Input Parameter	Data Type	Description
SPOper	REAL	SP operator value, scaled in PV units. SP is set to this value when in Operator control and not Cascade/Ratio mode. If the value of SPOper < SPLLimit or > SPHLimit, the instruction sets the appropriate bit in Status and limits the value used for SP. Valid = SPLLimit to SPHLimit Default = 0.0
SPCascade	REAL	SP Cascade value, scaled in PV units. If CascadeRatio is true and UseRatio is false, then SP = SPCascade. This is typically the CVEU of a primary loop. If CascadeRatio and UseRatio are true, then SP = (SPCascade x Ratio). If the value of SPCascade < SPLLimit or > SPHLimit, set the appropriate bit in Status and limit the value used for SP. Valid = SPLLimit to SPHLimit Default = 0.0
SPHLimit	REAL	SP high limit value, scaled in PV units. If SPHLimit > PVEUMax, the instruction sets the appropriate bit in Status. Valid = SPLLimit to PVEUMax Default = 100.0
SPLLimit	REAL	SP low limit value, scaled in PV units. If SPLLimit < PVEUMin, the instruction sets the appropriate bit in Status. If SPHLimit < SPLLimit, the instruction sets the appropriate bit in Status and limits SP using the value of SPLLimit. Valid = PVEUMin to SPHLimit Default = 0.0
UseRatio	BOOL	Allow ratio control permissive. Set to true to enable ratio control when in Cascade/Ratio mode. Default is false.
RatioProg	REAL	Ratio program multiplier. Ratio and RatioOper are set equal to this value when in Program control. If RatioProg < RatioLLimit or > RatioHLimit, the instruction sets the appropriate bit in Status and limits the value used for Ratio.

Input Parameter	Data Type	Description
		Valid = RatioLLimit to RatioHLimit Default = 1.0
RatioOper	REAL	Ratio operator multiplier. Ratio is set equal to this value when in Operator control. If RatioOper < RatioLLimit or > RatioHLimit, the instruction sets the appropriate bit in Status and limits the value used for Ratio. Valid = RatioLLimit to RatioHLimit Default = 1.0
RatioHLimit	REAL	Ratio high limit value. Limits the value of Ratio obtained from RatioProg or RatioOper. If RatioHLimit < RatioLLimit, the instruction sets the appropriate bit in Status and limits Ratio using the value of RatioLLimit. Valid = RatioLLimit to maximum positive float Default = 1.0
RatioLLimit	REAL	Ratio low limit value. Limits the value of Ratio obtained from RatioProg or RatioOper. If RatioLLimit < 0, the instruction sets the appropriate bit in Status and limits the value to zero. If RatioHLimit < RatioLLimit, the instruction sets the appropriate bit in Status and limits Ratio using the value of RatioLLimit. Valid = 0.0 to RatioHLimit Default = 1.0
CVFault	BOOL	Control variable bad health indicator. If CVEU controls an analog output, then CVFault normally comes from the analog output's fault status. When true, CVFault indicates an error on the output module and the instruction sets the appropriate bit in Status. Default is false = "good health"
CVInitReq	BOOL	CV initialization request. This signal is normally controlled by the "In Hold" status on the analog output module controlled by CVEU or from the InitPrimary output of a secondary PID loop. Default is false.

Input Parameter	Data Type	Description
CVInitValue	REAL	CVEU initialization value, scaled in CVEU units. When CVInitializing is true, CVEU = CVInitValue and CV equals the corresponding percentage value. CVInitValue comes from the feedback of the analog output controlled by CVEU or from the setpoint of a secondary loop. Instruction initialization is disabled when CVFaulted or CVEUSpanInv is true. Valid = any float Default = 0.0
CVProg	REAL	CV program manual value. CV equals this value when in Program Manual mode. If CVProg < 0 or > 100, or < CVLLimit or > CVHLimit when CVManLimiting is true, the instruction sets the appropriate bit in Status and limits the CV value. Valid = 0.0 to 100.0 Default = 0.0
CVOper	REAL	CV operator manual value. CV equals this value when in Operator Manual mode. If not Operator Manual mode, the instruction sets CVOper = CV at the end of each instruction execution. If CVOper < 0 or > 100, or < CVLLimit or > CVHLimit when CVManLimiting is true, the instruction sets the appropriate bit in Status and limits the CV value. Valid = 0.0 to 100.0 Default = 0.0
CVOverride	REAL	CV override value. CV equals this value when in override mode. This value should correspond to a safe state output of the PID loop. If CVOverride < 0 or >100, the instruction sets the appropriate bit in Status and limits the CV value. Valid = 0.0 to 100.0 Default = 0.0
CVPrevious	REAL	CV_{n-1} value. If CVSetPrevious is set, CV_{n-1} equals this value. CV_{n-1} is the value of CV from the previous execution. CVPrevious is ignored when in manual, override or hand

Input Parameter	Data Type	Description
		mode or when CVInitializing is set. If CVPrevious < 0 or > 100, or < CVLLimit or > CVHLimit when in Auto or cascade/ratio mode, the instruction sets the appropriate bit in Status and limits the CVn-1 value. Valid = 0.0 to 100.0 Default = 0.0
CVSetPrevious	BOOL	Request to use CVPrevious. If true, CVn-1 = CVPrevious Default is false.
CVManLimiting	BOOL	Limit CV in manual mode request. If Manual mode and CVManLimiting is true, CV is limited by the CVHLimit and CVLLimit values. Default is false.
CVEUMax	REAL	Maximum value for CVEU. The value of CVEU which corresponds to 100 percent CV. If CVEUMax = CVEUMin, the instruction sets the appropriate bit in Status. Valid = any float Default = 100.0
CVEUMin	REAL	Minimum value of CVEU. The value of CVEU which corresponds to 0 percent CV. If CVEUMax = CVEUMin, the instruction sets the appropriate bit in Status. Valid = any float Default = 0.0
CVHLimit	REAL	CV high limit value. This is used to set the CVHAlarm output. It is also used for limiting CV when in Auto or Cascade/Ratio mode, or Manual mode if CVManLimiting is true. If CVHLimit > 100 or < CVLLimit, the instruction sets the appropriate bit in Status. If CVHLimit < CVLLimit, the instruction limits CV using the value of CVLLimit. Valid = CVLLimit < CVHLimit ≤ 100.0 Default = 100.0
CVLLimit	REAL	CV low limit value. This is used to set the CVLAlarm output. It is also used for limiting CV when in Auto

Input Parameter	Data Type	Description
		or Cascade/Ratio mode, or Manual mode if CVManLimiting is true. If CVLLimit < 0 or CVHLimit < CVLLimit, the instruction sets the appropriate bit in Status. If CVHLimit < CVLLimit, the instruction limits CV using the value of CVLLimit. Valid = 0.0 ≤ CVLLimit < CVHLimit Default = 0.0
CVROCLimit	REAL	CV rate of change limit, in percent per second. Rate of change limiting is only used when in Auto or Cascade/Ratio modes or Manual mode if CVManLimiting is true. Enter 0 to disable CV ROC limiting. If CVROCLimit < 0, the instruction sets the appropriate bit in Status and disables CV ROC limiting. Valid = 0.0 to maximum positive float Default = 0.0
FF	REAL	Feed forward value. The value of feed forward is summed with CV after the zero-crossing deadband limiting has been applied to CV. Therefore changes in FF are always reflected in the final output value of CV. If FF < -100 or > 100, the instruction sets the appropriate bit in Status and limits the value used for FF. Valid = -100.0 to 100.0 Default = 0.0
FFPrevious	REAL	FF_{n-1} value. If FF SetPrevous is set, the instruction sets FF_{n-1} = FFPrevious. FF_{n-1} is the valu eof FF from the previous execution. If FFPrevious < -100 or > 100, the instruction sets the appropriate bit in Status and limits value used for FF_{n-1} Valid = -100.0 to 100.0 Default - 0.0
FFSetPrevious	BOOL	Request to use FFPrevious. If true, FFn-1 = FFPrevious. Default is false.

Input Parameter	Data Type	Description
HandFB	REAL	CV Hand feedback value. CV equals this value when in Hand mode and HandFBFault is false (good health). This value typically comes from the output of a field mounted hand/auto station and is used to generate a bumpless transfer out of hand mode. If HandFB < 0 or > 100, the instruction sets the appropriate bit in Status and limits the value used for CV. Valid = 0.0 to 100.0 Default = 0.0
HandFBFault	BOOL	HandFB value bad health indicator. If the HandFB value is read from an analog input, then HandFBFault is typically controlled by the status of the analog input channel. When true, HandFBFault indicates an error on the input module and the instruction sets the appropriate bit in Status. Default is false = "good health"
WindupHIn	BOOL	Windup high request. When true, the CV cannot integrate in a positive direction. The signal is typically obtained from the WindupHOut output from a secondary loop. Default is false.
WindupLIn	BOOL	Windup low request. When true, the CV cannot integrate in a negative direction. This signal is typically obtained from the WindupLOut output from a secondary loop. Default is false.
ControlAction	BOOL	Control action request. Set to true to calculate error as E = PV - SP; clear to false to calculate error as E = SP - PV. Default is false.
DependIndepend	BOOL	Dependent/independent control request. When true, use the dependent form of the PID equation; when false, use the independent form of the equations. Default is false.
PGain	REAL	Proportional gain. When the independent form of the PID

Input Parameter	Data Type	Description
		algorithm is selected, enter the unitless proportional gain into this value. When the dependent PID algorithm is selected, enter the unitless controller gain into this value. Enter 0 to disable the proportional control. If PGain < 0, the instruction sets the appropriate bit in Status and uses a value of PGain = 0. Valid = 0.0 to maximum positive float Default = 0.0
IGain	REAL	Integral gain. When the independent form of the PID algorithm is selected, enter the integral gain in units of 1/minutes into this value. When the dependent PID algorithm is selected, enter the integral time constant in units of minutes/repeat into this value. Enter 0 to disable the integral control. If IGain < 0, the instruction sets the appropriate bit in Status and uses a value of IGain = 0. Valid = 0.0 to maximum positive float Default = 0.0
DGain	REAL	Derivative gain. When the independent form of the PID algorithm is selected, enter the derivative gain in units of minutes into this value. When the dependent PID algorithm is used, enter the derivative time constant in units of minutes into this value. Enter 0 to disable the derivative control. If DGain < 0, the instruction sets the appropriate bit in Status and uses a value of DGain = 0. Valid = 0.0 to maximum positive float Default = 0.0
PVEProportional	BOOL	Proportional PV control request. When true, calculate the proportional term (DeltaPTerm) using the change in process variable (PVPercent). When false, use the change in error (EPercent). Default is false.
PVEDerivative	BOOL	Derivative PV control request. When true, calculate the derivative term (DeltaDTerm) using the change

Input Parameter	Data Type	Description
		in process variable (PVPercent). When false, use the change in error (EPercent). Default is true.
DSmoothing	BOOL	Derivative Smoothing request. When true, changes in the derivative term are smoothed. Derivative smoothing causes less output "jitters" as a result of a noisy PV signal but also limits the effectiveness of high derivative gains. Default is false.
PVTracking	BOOL	SP track PV request. Set to true to cause SP to track PV when in manual mode. Ignored when in Cascade/Ratio or Auto mode. Default is false.
ZCDeadband	REAL	Zero crossing deadband range, scaled in PV units. Defines the zero crossing deadband range. Enter 0 to disable the zero crossing deadband checking. If ZCDeadband < 0, the instruction sets the appropriate bit in Status and disables zero crossing deadband checking. Valid = 0.0 to maximum positive float Default = 0.0
ZCOff	BOOL	Zero crossing disable request. Set to true to disable zero crossing for the deadband calculation. Default is false.
PVHHLimit	REAL	PV high-high alarm limit value, scaled in PV units. Valid = any float Default = maximum positive float
PVHLimit	REAL	PV high alarm limit value, scaled in PV units. Valid = any float Default = maximum positive float
PVLLimit	REAL	PV low alarm limit value, scaled in PV units. Valid = any float Default = maximum negative float
PVLLLimit	REAL	PV low-low alarm limit value, scaled in PV units. Valid = any float

Input Parameter	Data Type	Description
		Default = maximum negative float
PVDeadband	REAL	PV alarm limit deadband value, scaled in PV units. Deadband is the delta value between the turn-on and turn-off value for each of the PV alarm limits. If PVDeadband < 0.0, the instruction sets the appropriate bit in Status and limits PVDeadband to zero. Valid = 0.0 to maximum positive float Default = 0.0
PVROCPosLimit	REAL	PV positive rate of change alarm limit. The limit value for a positive (increasing) change in PV, scaled in PV units per seconds. Enter 0.0 to disable positive PVROC alarm checking. If PVROCPosLimit < 0.0, the instruction sets the appropriate bit in Status and disables positive PVROC checking. Valid = 0.0 to maximum positive float Default = 0.0 PV/second
PVROCNegLimit	REAL	PV negative rate of change alarm limit. The limit value for a negative (decreasing) change in PV, scaled in PV units per seconds. Enter 0.0 to disable negative PVROC alarm checking. If PVROCNegLimit < 0, the instruction sets the appropriate bit in Status and disables negative PVROC checking. Valid = 0.0 to maximum positive float Default = 0.0
PVROCPeriod	REAL	PV rate of change sample period. The time period, in seconds, over which the rate of change for PV is evaluated. Enter 0 to disable PVROC alarm checking If PVROCPeriod < 0.0, the instruction sets the appropriate bit in Status, and disables positive and negative PVROC checking. Valid = any float ≥ 0.0 Default = 0.0 seconds
DevHHLimit	REAL	Deviation high-high alarm limit value, scaled in PV units. Deviation is the difference in value between the process variable (PV) and the setpoint (SP). Deviation alarming

Input Parameter	Data Type	Description
		alerts the operator to a discrepancy between the process variable and the setpoint value. If DevHHLimit < 0.0, the instruction sets the appropriate bits in Status and sets DevHHLimit = 0.0. Valid = 0.0 to maximum positive float Default = maximum positive float
DevHLimit	REAL	Deviation high alarm limit value, scaled in PV units. Deviation is the difference in value between the process variable (PV) and the setpoint (SP). Deviation alarming alerts the operator to a discrepancy between the process variable and the setpoint value. If DevHLimit < 0.0, the instruction sets the appropriate bit in Status and sets DevHLimit = 0.0. Valid = 0.0 to maximum positive float Default = maximum positive float
DevLLimit	REAL	Deviation low alarm limit value, scaled in PV units. Deviation is the difference in value between the process variable (PV) and the setpoint (SP). Deviation alarming alerts the operator to a discrepancy between the process variable and the setpoint value. If DevLLimit < 0.0, the instruction sets the appropriate bit in Status and sets DevLLimit = 0.0. Valid = 0.0 to maximum positive float Default = maximum positive float
DevLLLimit	REAL	Deviation low-low alarm limit value, scaled in PV units. Deviation is the difference in value between the process variable (PV) and the setpoint (SP). Deviation alarming alerts the operator to a discrepancy between the process variable and the setpoint value. If DevLLLimit < 0.0, the instruction sets the appropriate bit in Status and sets DevLLLimit = 0.0. Valid = 0.0 to maximum positive float Default = maximum positive float
DevDeadband	REAL	The deadband value for the Deviation alarm limits, scaled in PV

Input Parameter	Data Type	Description
		units. Deadband is the delta value between the turn-on and turn-off value for each of the Deviation alarm limits. If DevDeadband < 0.0, the instruction sets the appropriate bit in Status and sets DevDeadband = 0.0. Valid = 0.0 to maximum positive float Default = 0.0
AllowCasRat	BOOL	Allow cascade/ratio mode permissive. Set to true to allow Cascade/Ratio mode to be selected using either ProgCasRatReq or OperCasRatReq. Default is false.
ManualAfterInit	BOOL	Manual mode after initialization request. When true, the instruction is placed in Manual mode when CVInitializing is true, unless the current mode is Override or Hand. When ManualAfterInit is false, the instruction's mode is not changed, unless requested to do so. Default is false.
ProgProgReq	BOOL	Program program request. Set to true by the user program to request Program control. Ignored if ProgOperReq is true. Holding this true and ProgOperReq false locks the instruction in Program control. When ProgValueReset is true, the instruction clears this input to false at each execution. Default is false.
ProgOperReq	BOOL	Program operator request. Set to true by the user program to request Operator control. Holding this true locks the instruction in Operator control. When ProgValueReset is true, the instruction clears this input to false at each execution. Default is false.
ProgCasRatReq	BOOL	Program cascade/ratio mode request. Set to true by the user program to request Cascade/Ratio mode. When ProgValueReset is true, the instruction clears this input to false at each execution.

Input Parameter	Data Type	Description
		Default is false.
ProgAutoReq	BOOL	Program auto mode request. Set to true by the user program to request Auto mode. When ProgValueReset is true, the instruction clears this input to false at each execution. Default is false.
ProgManualReq	BOOL	Program manual mode request. Set to true by the user program to request Manual mode. When ProgValueReset is true, the instruction clears this input to false ateach execution. Default is false.
ProgOverrideReq	BOOL	Program override mode request. Set to true by the user program to request Override mode. When ProgValueReset is true, the instruction clears this input to false at each execution. Default is false.
ProgHandReq	BOOL	Program hand mode request. Set to true by the user program to request Hand mode. This value is usually read as a digital input from a hand/auto station. When ProgValueReset is true, the instruction clears this input to false at each execution. Default is false.
OperProgReq	BOOL	Operator program request. Set to true by the operator interface to request Program control. The instruction clears this input to false at each execution. Default is false.
OperOperReq	BOOL	Operator operator request. Set to true by the operator interface to request Operator control. The instruction clears this input to false at each execution. Default is false.
OperCasRatReq	BOOL	Operator cascade/ratio mode request. Set to true by the operator interface to request Cascade/ Ratio mode. The instruction clears this input to false at each execution.

Input Parameter	Data Type	Description
		Default is false.
OperAutoReq	BOOL	Operator auto mode request. Set to true by the operator interface to request Auto mode. The instruction clears this input to false at each execution. Default is false.
OperManualReq	BOOL	Operator manual mode request. Set to true by the operator interface to request Manual mode. The instruction clears this input to false at each execution. Default is false.
ProgValueReset	BOOL	Reset program control values. When true, all the program request inputs are cleared to false by the instruction at each execution. When true and in Operator control, the instruction sets SPProgram = SP and CVProgram = CV. Default is false.
TimingMode	DINT	Selects timing execution mode. 0 = Periodic mode 1 = Oversample mode 2 = Real time sampling mode For more information about timing modes, see Function Block Attributes. Valid = 0 to 2 Default = 0
OversampleDT	REAL	Execution time for oversample mode. Valid = 0 to 4194.303 seconds Default = 0
RTSTime	DINT	Module update period for real time sampling mode Valid = 1 to 32,767ms Default = 1
RTSTimeStamp	DINT	Module time stamp value for real time sampling mode. Valid = 0 to 32,767ms Default = 0
AtuneAcquire	BOOL	Acquire PIDE AtuneData request. Default is false.

Input Parameter	Data Type	Description
AtuneStart	BOOL	Start Autotune request. Default is false.
AtuneUseGains	BOOL	Use Autotune gains request. Default is false.
AtuneAbort	BOOL	Abort Autotune request. Default is false.
AtuneUnacquire	BOOL	Unacquire PIDE AtuneData request. Default is false.

Output Parameter	Data Type	Description
EnableOut	BOOL	Indicates if instruction is enabled. Cleared to false if CVEU overflows.
CVEU	REAL	Scaled control variable output. Scaled using CVEUMax and CVEUMin, where CVEUMax corresponds to 100 percent and CVEUMin corresponds to 0 percent. This output typically controls an analog output module or a secondary loop. CVEU = (CV x CVEUSpan / 100) + CVEUMin CVEU span calculation: CVEUSpan = (CVEUMax - CVEUMin)
CV	REAL	Control variable output. This value is expressed as 0 to 100 percent. CV is limited by CVHLimit and CVLLimit when in auto or cascade/ratio mode or manual mode if CVManLimiting is true. Otherwise this value is limited by 0 and 100 percent.
CVInitializing	BOOL	Initialization mode indicator. CVInitializing is set to true when CVInitReq is true, during instruction first scan, and on a true to false transition of CVHealth (bad to good). CVInitializing is cleared to false after the instruction has been initialized and CVInitReq is false.
CVHAlarm	BOOL	CV high alarm indicator. Set to true when the calculated value of

Output Parameter	Data Type	Description
		CV > 100 or CVHLimit.
CVLAlarm	BOOL	CV low alarm indicator. Set to true when the calculated value of CV < 0 or CVLLimit.
CVROCAlarm	BOOL	CV rate of change alarm indicator. Set to true when the calculated rate of change for CV exceeds CVROCLimit.
SP	REAL	Current setpoint value. The value of SP is used to control CV when in Auto or Cascade/ Ratio mode.
SPPercent	REAL	The value of SP expressed in percent of span of PV. SPPercent = ((SP - PVEUMin) x 100) / PVSpan PV Span calculation: PVSpan = (PVEUMax - PVEUMin)
SPHAlarm	BOOL	SP high alarm indicator. Set to true when the SP > SPHLimit.
SPLAlarm	BOOL	SP low alarm indicator. Set to true when the SP < SPLLimit.
PVPercent	REAL	PV expressed in percent of span. PVPercent = ((PV- PVEUMin) x 100) / PVSpan PV Span calculation: PVSpan = (PVEUMax - PVEUMin)
E	REAL	Process error. Difference between SP and PV, scaled in PV units.
EPercent	REAL	The error expressed as a percent of span.
InitPrimary	BOOL	Initialize primary loop command. Set to true when not in Cascade/Ratio mode or when CVInitializing is true. This signal is normally used by the CVInitReq input of a primary PID loop.
WindupHOut	BOOL	Windup high indicator. Set to true when CV high or CV low limit (depending on the control action) or SP high limit has been reached. This signal is typically used by the WindupHIn input to prevent the

Output Parameter	Data Type	Description
		windup of the CV output on a primary loop.
WindupLOut	BOOL	Windup low indicator. Set to true when CV high or CV low limit (depending on the control action) or SP low limit has been reached. This signal is typically used by the WindupLIn input to prevent the windup of the CV output on a primary loop.
Ratio	REAL	Current ratio multiplier.
RatioHAlarm	BOOL	Ratio high alarm indicator. Set to true when Ratio > RatioHLimit.
RatioLAlarm	BOOL	Ratio low alarm indicator. Set to true when Ratio < RatioLLimit.
ZCDeadbandOn	BOOL	Zero crossing deadband indicator. When true the value of CV does not change. If ZCOff is true, then ZCDeadbandOn is set to true when | E | is within the ZCDeadband range. If ZCOff is false, then ZCDeadbandOn is set to true when | E | crosses zero and remains within the ZCDeadband range. ZCDeadbandOn is cleared to false when | E | exceeds the deadband range or when ZCDeadband = 0.
PVHHAlarm	BOOL	PV high-high alarm indicator. Set to true when PV ≥ PVHHLimit. Cleared to false when PV < (PVHHLimit - PVDeadband)
PVHAlarm	BOOL	PV high alarm indicator. Set to true when PV ≥ PVHLimit. Cleared to false when PV < (PVHLimit - PVDeadband)
PVLAlarm	BOOL	PV low alarm indicator. Set to true when PV ≤ PVLLimit. Cleared to false when PV > (PVLLimit + PVDeadband)

Output Parameter	Data Type	Description
PVLLAlarm	BOOL	PV low-low alarm indicator. Set to true when PV ≤ PVLLLimit. Cleared to false when PV > (PVLLLimit + PVDeadband)
PVROCPosAlarm	BOOL	PV positive rate-of-change alarm indicator. Set to true when calculated PV rate-of-change ≥ PVROCPosLimit.
PVROCNegAlarm	BOOL	PV negative rate-of-change alarm indicator. Set to true when calculated PV rate-of-change ≤ (PVROCNegLimit x -1).
DevHHAlarm	BOOL	Deviation high-high alarm indicator. Set to true when PV ≥ (SP + DevHHLimit). Cleared to false when PV < (SP + DevHHLimit - DevDeadband)
DevHAlarm	BOOL	Deviation high alarm indicator. Set to true when PV ≥ (SP + DevHLimit). Cleared to false when PV < (SP + DevHLimit - DevDeadband)
DevLAlarm	BOOL	Deviation low alarm indicator. Set to true when PV ≤ (SP - DevLLimit). Cleared to false when PV > (SP - DevLLimit + DevDeadband)
DevLLAlarm	BOOL	Deviation low-low alarm indicator. Set to true when PV ≤ (SP - DevLLLimit). Cleared to false when PV > (SP - DevLLLimit + DevDeadband)
ProgOper	BOOL	Program/operator control indicator. Set to true when in Program control. Cleared to false when in Operator control.

Output Parameter	Data Type	Description
CasRat	BOOL	Cascade/ratio mode indicator. Set to true when in the Cascade/Ratio mode.
Auto	BOOL	Auto mode indicator. Set to true when in the Auto mode.
Manual	BOOL	Manual mode indicator. Set to true when in the Manual mode.
Override	BOOL	Override mode indicator. Set to true when in the Override mode.
Hand	BOOL	Hand mode indicator. Set to true when in the Hand mode.
DeltaT	REAL	Elapsed time between updates. This is the elapsed time in seconds used by the control algorithm to calculate the process output.
AtuneReady	BOOL	Set to true when the specified AtuneData has been acquired by the PIDE instruction.
AtuneOn	BOOL	Set to true when auto tuning has been initiated.
AtuneDone	BOOL	Set to true when auto tuning has completed.
AtuneAborted	BOOL	Set to true when auto tuning has been aborted by the user or due to errors that occurred during the auto tuning operation.
AtuneBusy	BOOL	Set to true when the specified AtuneData could not be acquired because it is currently acquired by another PIDE instruction.
Status1	DINT	Status of the function block.
InstructFault (Status1.0)	BOOL	The instruction detected one of the following execution errors. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.
PVFaulted (Status1.1)	BOOL	Process variable (PV) health bad.
CVFaulted (Status1.2)	BOOL	Control variable (CV) health bad.

Output Parameter	Data Type	Description
HandFBFaulted (Status1.3)	BOOL	HandFB value health bad.
PVSpanInv (Status1.4)	BOOL	Invalid span of PV. PVEUMax £ PVEUMin.
SPProgInv (Status1.5)	BOOL	SPProg < SPLLimit or SPProg > SPHLimit. The instruction uses the limited value for SP.
SPOperInv (Status1.6)	BOOL	SPOper < SPLLimit or SPOper > SPHLimit. The instruction uses the limited value for SP.
SPCascadeInv (Status1.7)	BOOL	SPCascade < SPLLimit or SPCascade > SPHLimit. The instruction uses the limited value for SP.
SPLimitsInv (Status1.8)	BOOL	Limits invalid: SPLLimit < PVEUMin, SPHLimit > PVEUMax, or SPHLimit < SPLLimit. If SPHLimit < SPLLimit, the instruction limits the value using SPLLimit
RatioProgInv (Status1.9)	BOOL	RatioProg < RatioLLimit or RatioProg > RatioHLimit. The instruction limits the value for Ratio.
RatioOperInv (Status1.10)	BOOL	RatioOper < RatioLLimit or RatioOper > RatioHLimit. The instruction limits the value for Ratio.
RatioLimitsInv (Status1.11)	BOOL	RatioLLimit < 0 or RatioHLimit < RatioLLimit.
CVProgInv (Status1.12)	BOOL	CVProg < 0 or CVProg > 100, or CVProg < CVLLimit or CVProg > CVHLimit when CVManLimiting is true. The instruction limits the value for CV.
CVOperInv (Status1.13)	BOOL	CVOper < 0 or CVOper > 100, or CVOper < CVLLimit or CVOper > CVHLimit when CVManLimiting is true. The instruction limits the value for CV.
CVOverrideInv (Status1.14)	BOOL	CVOverride < 0 or CVOverride > 100. The instruction limits the value for CV.
CVPreviousInv (Status1.15)	BOOL	CVPrevious < 0 or CVPrevious > 100, or CVPrevious < CVLLimit or

Output Parameter	Data Type	Description
		CVPrevious > CVHLimit when in Auto or Cascade/Ratio mode. The instruction limits the value of CVn-1.
CVEUSpanInv (Status1.16)	BOOL	Invalid CVEU span. The instruction uses a value of CVEUMax = CVEUMin.
CVLimitsInv (Status1.17)	BOOL	CVLLimit < 0, CVHLimit > 100, or CVHLimit < CVLLimit. If CVHLimit < CVLLimit, the instruction limits CV using CVLLimit.
CVROCLimitInv (Status1.18)	BOOL	CVROCLimit < 0. The instruction disables ROC limiting.
FFInv (Status1.19)	BOOL	FF < -100 or FF > 100. The instruction uses the limited value for FF.
FFPreviousInv (Status1.20)	BOOL	FFPrevious < - 100 or FFPrevious > 100. The instruction uses the limited value for FF_{n-1}.
HandFBInv (Status1.21)	BOOL	HandFB < 0 or HandFB > 100. The instruction uses the limited value for CV.
PGainInv (Status1.22)	BOOL	PGain < 0. The instruction uses a value of PGain = 0.
IGainInv (Status1.23)	BOOL	IGain < 0. The instruction uses a value of IGain = 0.
DGainInv (Status1.24)	BOOL	DGain < 0. The instruction uses a value of DGain = 0.
ZCDeadbandInv (Status1.25)	BOOL	ZCDeadband < 0. The instruction disables zero crossing deadband.
PVDeadbandInv (Status1.26)	BOOL	PVDeadband < 0. The instruction limits PVDeadband to zero.
PVROCLimitsInv (Status1.27)	BOOL	PVROCPosLimit < 0, PVROCNegLimit < 0, or PVROCPeriod < 0.
DevHLLimitsInv (Status1.28)	BOOL	Deviation high-low limits invalid. Low-low limit < 0, low limit < 0, high limit < 0, or high-high limit < 0. The instruction uses 0 for the invalid limit.

Output Parameter	Data Type	Description
DevDeadbandInv (Status1.29)	BOOL	Deviation deadband < 0. The instruction uses a value of DevDeadband = 0.
Status2	DINT	Timing status of the function block.
TimingModeInv (Status2.27)	BOOL	Invalid TimingMode value. For more information about timing modes, see Function Block Attributes.
RTSMissed (Status2.28)	BOOL	Only used in real time sampling mode. Set to true when ABS | DeltaT - RTSTime | > 1 (.001 second).
RTSTimeInv (Status2.29)	BOOL	Invalid RTSTime value.
RTSTimeStampInv (Status2.30)	BOOL	Invalid RTSTimeStamp value.
DeltaTInv (Status2.31)	BOOL	Invalid DeltaT value.

Description

The PID algorithm regulates the CV output in order to maintain the PV at the SP when the instruction executes in Cascade/Ratio or Auto modes.

When ControlAction is set, the calculated value of EPercent and PVPIDPercent is negated before being used by the control algorithm.

The following table describes how the instruction calculates the PID terms.

PID Term	Method of Calculation
proportional	The proportional term is calculated using: • PV when PVEProportional is set or • Error when PVEProportional is cleared Set PGain = 0 to disable proportional control.
integral	The integral term is calculated using Error. Set IGain = 0 to disable integral control. Also, setting PGain = 0 when DependIndepend is set will disable integral control.
derivative	The derivative term is calculated using: • PV when PVEDerivative is set or • Error when PVEDerivative is cleared Set DGain = 0 to disable derivative control. Also, setting PGain = 0 when DependIndepend is set will disable derivative control.

PID Term	Method of Calculation
	Derivative smoothing is enabled when DSmoothing is set and disabled when DSmoothing is cleared. Derivative smoothing causes less CV output "jitter" as a result of a noisy PV signal but also limits the effectiveness of high derivative gains.

Computing CV

The PID control algorithm computes the value for CV by summing Delta PTerm, Delta ITerm, Delta

DTerm, and CV from the previous execution of the instruction, for example CV_{n-1}. When CVsetPrevious is set,

CVn-1 is set equal to CVPrevious. This lets you preset CV_{n-1} to a specified value before computing the value of CV.

CalculatedCV = CV_{n-1} + DΔPTerm + DITerm + DDTerm

PIDE Algorithms

The PIDE instruction uses a velocity form PID algorithm similar to that used in most DCS systems. Some advantages to a velocity form algorithm include:

- Bumpless adaptive gain changes – You can change gains on the fly without initializing the algorithm.
- Multi-loop control schemes – You can implement cross limiting between loops by manipulating the CV_{n-1} term.

Independent Gains Form

$$CV_n = CV_{n-1} + K_P\Delta E + \frac{K_I}{60}E\Delta t + 60K_D\frac{E_n - 2E_{n-1} + E_{n-2}}{\Delta t}$$

In this form of the algorithm, each term of the algorithm (proportional, integral, and derivative) has a separate gain. Changing one gain only affects that term and not any of the others, where:

PIDE term:	Description:
CV	Control variable
E	Error in percent of span
Dt	Update time in seconds used by the loop
K_p	Proportional gain
K_I	Integral gain in min $^{-1}$ a larger value of K_I causes a faster integral response.
K_D	Derivative gain in minutes

Dependent Gains Form

$$CV_n = CV_{n-1} + K_C\left(\Delta E + \frac{1}{60T_I}E\Delta t + 60T_D\frac{E_n - 2E_{n-1} + E_{n-2}}{\Delta t}\right)$$

This form of the algorithm changes the proportional gain into a controller gain. By changing the controller gain, you change the action of all three terms (proportional, integral, and derivative) at the same time, where:

PIDE term:	Description:
CV	Control variable
E	Error in percent of span
Dt	Update time in seconds used by the loop
K_c	Controller gain
T_I	Integral time constant in minutes per repeat a larger value of T_1 causes a slower integral response It takes T_1 minutes for the integral term to repeat the action of the proportional term in response to a step change in error.
T_D	Derivative time in constant in minutes

PIDE term:	Description:
CV	Control variable
E	Error in percent of span
Dt	Update time in seconds used by the loop
Kp	Proportional gain
K_I	Integral gain in min $^{-1}$ a larger value of K_I causes a faster integral response.
K_D	Derivative gain in minutes

Determining Which Algorithm to Use

- $K_P = K_C$
- $K_I = \frac{K_C}{T_I}$
- $K_D = K_C T_D$

The PIDE equations above are representative of the algorithms used by the PIDE instruction. You can substitute the change in error values for the change in PV (in percent of span) for the proportional and derivative terms by manipulating the parameters PVEProportional and PVEDerivative. By default, the PIDE instruction uses the change in error for the proportional term and the change in PV for the derivative term. This eliminates large derivative spikes on changes in setpoint.

You can convert the gains used between the different PIDE algorithm forms using these equations:

- $K_P = K_C$
- $K_I = \frac{K_C}{T_I}$
- $K_D = K_C T_D$

Each algorithm provides identical control with the appropriate gains. Some people prefer the independent gains style because they can manipulate individual gains without affecting the other terms. Others prefer the dependent gains style because they can, at least to a certain extent, change just the controller gain and cause an overall change in the aggressiveness of the PID loop without changing each gain separately.

Monitoring the PIDE Instruction

There is an operator faceplate available for the PIDE instruction.

Autotuning the PIDE Instruction

The Logix DesignerLogix Designer application PIDE autotuner provides an open-loop autotuner built into the PIDE instruction. You can autotune from PanelView terminal or any other operator interface devices, as well as the Logix Designer application. The PIDE block has an Autotune Tag (type PIDE_AUTOTUNE) that you specify for those PIDE blocks that you want to autotune.

The PIDE autotuner is installed with the application, but you need an activation key to enable it. The autotuner is only supported in function block programming; it is not available in ladder diagram or structured text programming.

Use the Autotune tab to specify and configure the autotune tag for a PIDE block.

Affects Math Status Flags

No

Major/Minor Faults

None specific to this instruction. See Common Attributes on page for operand-related faults.

Execution

Function Block

Condition/State	Action Taken
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is true	EnableIn and EnableOut bits are set to true. The instruction executes.
Instruction first run	N/A
Instruction first scan	If CVFault and CVEUSpanInv are set,see Processing Faults later in this instruction. If CVFault and CVEUSpanInv are cleared: 1. CVInitializing is set 2. If PVFault is set, PVSpanInv and SPlimitsInv are cleared. See Processing Faults in this instruction. 3. The PID control algorithm is not executed. 4. The instruction sets CVEU = CVInitValue and CV = corresponding percentage. 5. When CVInitializing and Manual AfterInit are set, the instructions misables auto and cascade/ratio modes. If the current mode is not Override or Hand mode, the instruction changes to manual ode.If ManualAfterInit is cleared, the mode is not changed. CVEu = CVInitValue CV_{n-1} = CV = CVEU - CVEUMin x100 CVEUMax - CVEUMin CVOper = CV
Postscan	EnableIn and EnableOut bits are cleared to false.

Structured Text

Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.
Normal Execution	See Tag.EnableIn is true in the Function Block table.
Postscan	See Postscan in the Function Block table.

When CVInitReq is set, or during instruction first scan, or on a set to cleared transition of CVFault (bad to good), the instruction initializes the CVEU and CV outputs to the value of CVInitValue. If the timing mode is not oversample and EnableIn transitions from cleared to

set, the instruction initializes the CVEU and CV values. CVInitialization is cleared after the initialization and when CVInitReq is cleared.

The CVInitValue normally comes from the analog output's readback value. The CVInitReq value normally comes from the "In Hold" status bit on the analog output controlled by CVEU. The initialization procedure is performed to avoid a bump at startup in the output signal being sent to the field device.

When using cascaded PID loops, the primary PID loop can be initialized when the secondary loop is initialized or when the secondary loop leaves the Cascade/Ratio mode. In this case, move the state of the InitPrimary output and SP output from the secondary loop to the CVInitReq input and CVInitValue input on the primary loop.

The instruction does not initialize and the CVEU and CV values are not updated if CVFault or CVEUSpanInv is set.

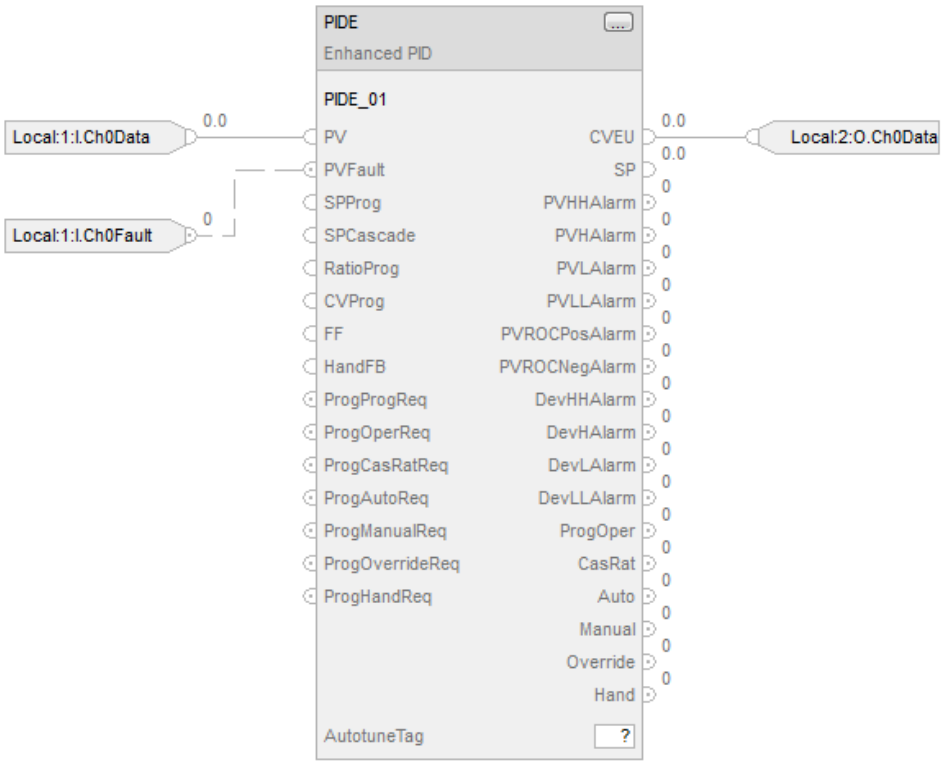
Examples

Example 1

The easiest way to implement a PIDE instruction is to create a function block routine in a program in a periodic task. The default timing mode for the PIDE instruction is periodic. When the PIDE instruction is used in a periodic task and in periodic timing mode, it automatically uses the periodic task's update rate as its delta t update time. All you need to do is wire the process variable analog input into the PV parameter on the PIDE instruction and wire the CVEU out of the PIDE instruction into the controlled variable analog output.

Optionally, you can wire the analog input's fault indicator (if one is available) into the PVFault parameter on the PIDE instruction. This forces the PIDE into Manual mode when the analog input is faulted and stops the PIDE CVEU output from winding up or down when the PV signal is not available.

Function Block



Structured Text

PIDE_01.PV := Local:1:I.Ch0Data;

PIDE_01.PVFault := Local:1:I.Ch0Fault;

PIDE(PIDE_01);

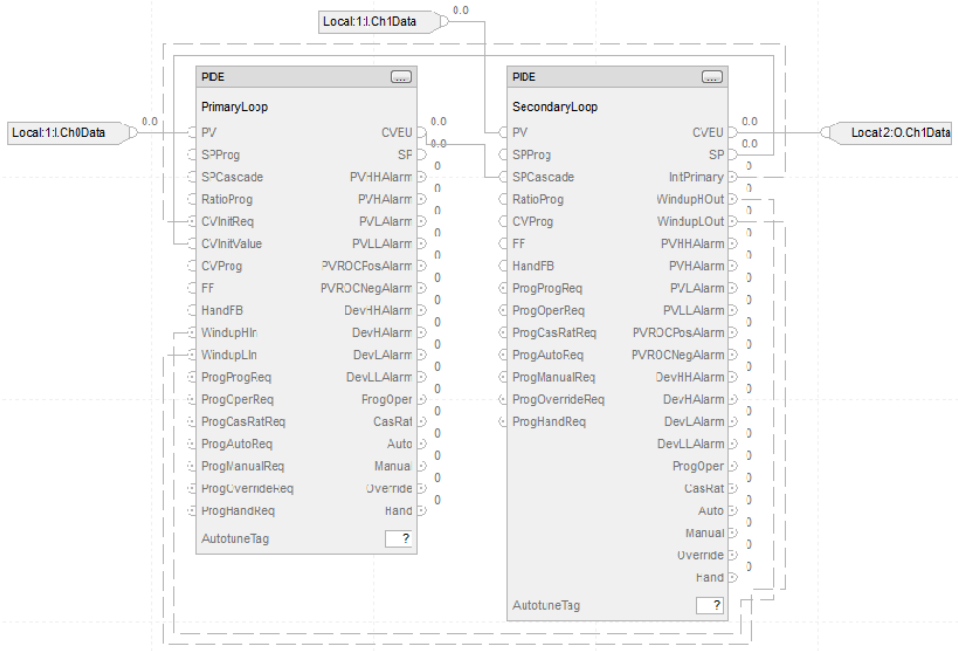
Local:2:)0.Ch0Data :=PIDE_01.CVEU;

Example 2

Cascade control is useful when externally-caused upsets to the controlled variable occur often, which then cause upsets to the process variable you are trying to control. For example, try to control the temperature of liquid in a tank by varying the amount of steam fed into a heating jacket around the tank. If the steam flow suddenly drops because of an upstream process, the temperature of the liquid in the tank eventually drops and the PIDE instruction then opens the steam valve to compensate for the drop in temperature.

In this example, a cascaded loop provides better control by opening the steam valve when the steam flow drops before the liquid temperature in the tank drops. To implement a cascaded loop, use a PIDE instruction to control the steam valve opening based on a process variable signal from a steam flow transmitter. This is the secondary loop of the cascaded pair. A second PIDE instruction (called the primary loop) uses the liquid temperature as a process variable and sends its CV output into the setpoint of the secondary loop. In this manner, the primary temperature loop asks for a certain amount of steam flow from the secondary steam flow loop. The steam flow loop is then responsible for providing the amount of steam requested by the temperature loop in order to maintain a constant liquid temperature.

Function Block



Structured Text

```
PrimaryLoop.PV := Local:1:I.CH0Data;
PrimaryLoop.CVInitReq := SecondaryLoop.InitPrimary;
PrimaryLoop.CVInitValue := SecondaryLoop.SP;
PrimaryLoop.WindupHIn := SecondaryLoop.WindupHOut;
PrimaryLoop.WindupLIn := SecondaryLoop.WindupLOut;

PIDE(PrimaryLoop);

SecondaryLoop.PV := Local:1:I.Ch1Data;
SecondaryLoop.SPCascade := PrimaryLoop.CVEU;

PIDE(SecondaryLoop);

Local:2:O.Ch0Data:= SecondaryLoop.CVEU;
```

For a cascaded pair of loops to work correctly, the secondary loop must have a faster process response than the primary loop. This is because the secondary loop's process must be able to compensate for any upsets before these upsets affect the primary loop's process. In this example, if steam flow drops, the steam flow must be able to increase as a result of the secondary controller's action before the liquid temperature is affected.

To set up a pair of cascaded PIDE instructions, set the *AllowCasRat* input parameter in the secondary loop. This allows the secondary loop to be placed into Cascade/Ratio mode. Next, wire the *CVEU* from the primary loop into the *SPCascade* parameter on the secondary loop. The *SPCascade* value is used as the SP on the secondary loop when the secondary loop is placed into Cascade/Ratio mode. The engineering unit range of the CVEU on the primary loop should match the engineering unit range of the PV on the secondary loop. This lets the primary loop scale its 0-100% value of CV into the matching engineering units used for the setpoint on the secondary loop.

The PIDE instruction supports several other features to more effectively support cascade control. Wire the *InitPrimary* output on the secondary loop into the *CVInitReq* input on the primary loop and wire the SP output of the secondary into the *CVInitValue* input on the primary.

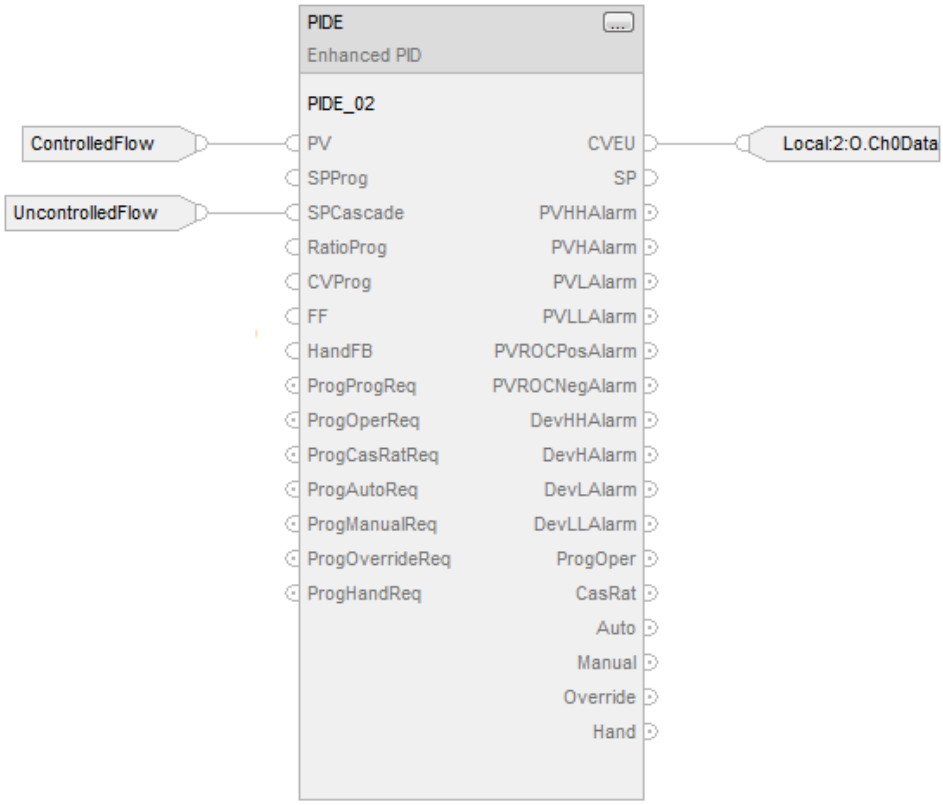
This sets the CVEU value of the primary loop equal to the SP of the secondary loop when the secondary loop leaves Cascade/Ratio mode. This allows a bumpless transfer when you place the secondary loop back into Cascade/Ratio mode. Also, wire the *WindupHOut* and WindupLOut outputs on the secondary loop into the *WindupHIn* and *WindupLIn* inputs on the primary loop. This causes the primary loop to stop increasing or decreasing, as appropriate, its value of CVEU if the secondary loop hits a SP limit or CV limit and eliminates any windup on the primary loop if these conditions occur.

Example 3

Ratio control is typically used to add a fluid in a set proportion to another fluid. For example, if you want to add two reactants (say A and B) to a tank in a constant ratio, and the flow rate of reactant A may change over time because of some upstream process upsets, you can use a ratio controller to automatically adjust the rate of reactant B addition. In this example, reactant A is often called the "uncontrolled" flow since it is not controlled by the PIDE instruction. Reactant B is then called the "controlled" flow.

To perform ratio control with a PIDE instruction, set the *AllowCasRat* and *UseRatio* input parameters. Wire the uncontrolled flow into the *SPCascade* input parameter. When in Cascade/Ratio mode, the uncontrolled flow is multiplied by either the *RatioOper* (when in Operator control) or the RatioProg (when in Program control) and the resulting value is used by the PIDE instruction as the setpoint.

Function Block



Structured Text

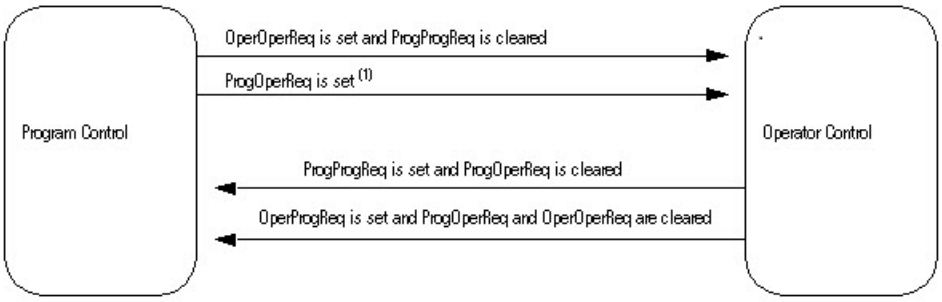
```
PIDE_01.PV := ControlledFlow;
PIDE_01.SPCascade := UncontrolledFlow;
```

```
PIDE(PIDE_01);

Local:2:0.Ch0Data := PIDE_01.CVEU;
```

Switching Between Program Control and Operator Control

The PIDE instruction can be controlled by either a user program or an operator interface. You can change the control mode at any time. Program and Operator control use the same ProgOper output. When ProgOper is set, control is Program; when ProgOper is cleared, control is Operator.

The following diagram shows how the PIDE instruction changes between Program control and Operator control.



(1) The instruction remains in Operator control mode when ProgOperReq is set.

Operating Modes

The PIDE instruction supports the following PID modes.

PID Operating Mode	Description
Cascade/Ratio	While in Cascade/Ratio mode the instruction computes the change in CV. The instruction regulates CV to maintain PV at either the SPCascade value or the SPCascade value multiplied by the Ratio value. SPCascade comes from either the CVEU of a primary PID loop for cascade control or from the "uncontrolled" flow of a ratio-controlled loop. Select Cascade/Ratio mode using either OperCasRatReq or ProgCasRatReq: Set OperCasRatReq to request Cascade/Ratio mode. Ignored when ProgOper, ProgOverrideReq, ProgHandReq, OperAutoReq, or OperManualReq is set, or when AllowCasRat is cleared. Set ProgCasRatReq to request Cascade/Ratio mode. Ignored when ProgOper or AllowCasRat is cleared or when ProgOverrideReq, ProgHandReq, ProgAutoReq, or ProgManualReq is set.
Auto	While in Auto mode the instruction computes the change in CV. The instruction regulates CV to maintain PV at the SP value. If in program control, SP = SPProg; if in Operator control, SP = SPOper.

PID Operating Mode	Description
	Select Auto mode using either OperAutoReq or ProgAutoReq: Set OperAutoReq to request Auto mode. Ignored when ProgOper, ProgOverrideReq, ProgHandReq, or OperManualReq is set. Set ProgAutoReq to request Auto mode. Ignored when ProgOper is cleared or when ProgOverrideReq, ProgHandReq, or ProgManualReq is set.
Manual	While in Manual mode the instruction does not compute the change in CV. The value of CV is determined by the control. If in Program control, CV = CVProg; if in Operator control, CV = CVOper. Select Manual mode using either OperManualReq or ProgManualReq: Set OperManualReq to request Manual mode. Ignored when ProgOper, ProgOverrideReq, or ProgHandReq is set. Set ProgManualReq to request Manual mode. Ignored when ProgOper is cleared or when ProgOverrideReq or ProgHandReq is set.
Override	While in Override mode the instruction does not compute the change in CV. CV = CVOverride, regardless of the control mode. Override mode is typically used to set a "safe state" for the PID loop. Select Override mode using ProgOverrideReq: Set ProgOverrideReq to request Override mode. Ignored when ProgHandReq is cleared.
Hand	While in Hand mode the PID algorithm does not compute the change in CV. CV = HandFB, regardless of the control mode. Hand mode is typically used to indicate that control of the final control element was taken over by a field hand/auto station. Select Hand mode using ProgHandReq: Set ProgHandReq to request hand mode. This value is usually read as a digital input from a hand/auto station.

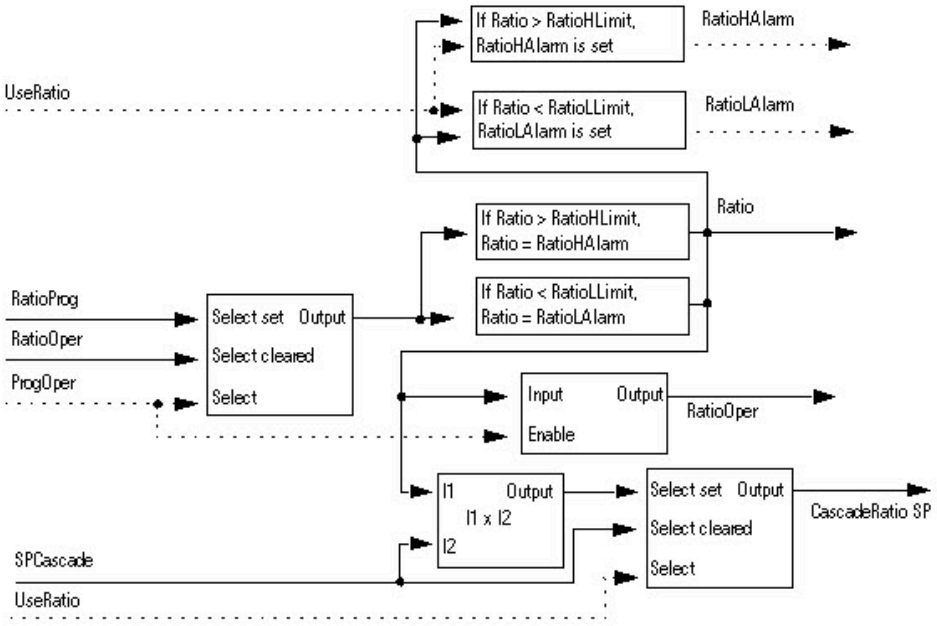
The Cascade/Ratio, Auto, and Manual modes can be controlled by a user program when in Program control or by an operator interface when in Operator control. The Override and Hand modes have a mode request input that can only be controlled by a user program; these inputs operate in both Program and Operator control.

Selecting the Setpoint

Once the instruction determines program or operator control and the PID mode, the instruction can obtain the proper SP value. You can select the cascade/ratio SP or the current SP.

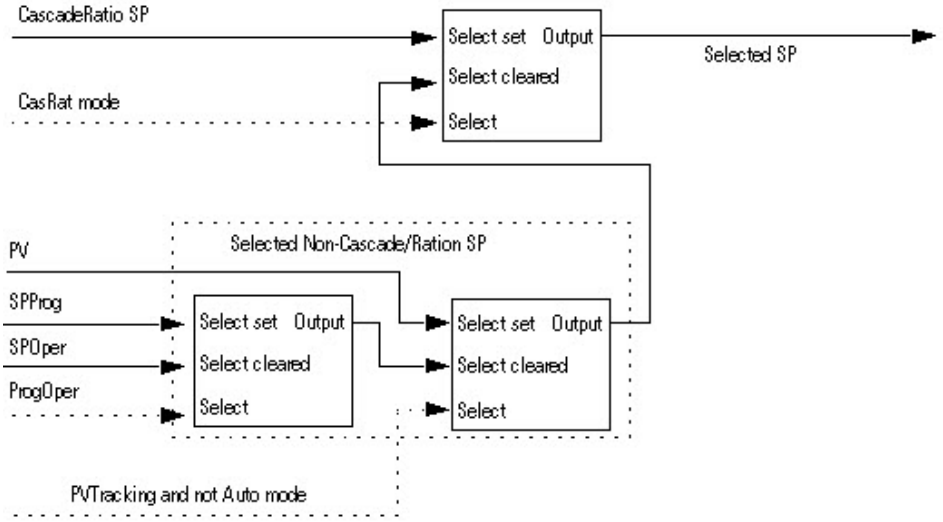
Cascade/Ratio SP

The cascade/ratio SP is based on the UseRatio and ProgOper values.



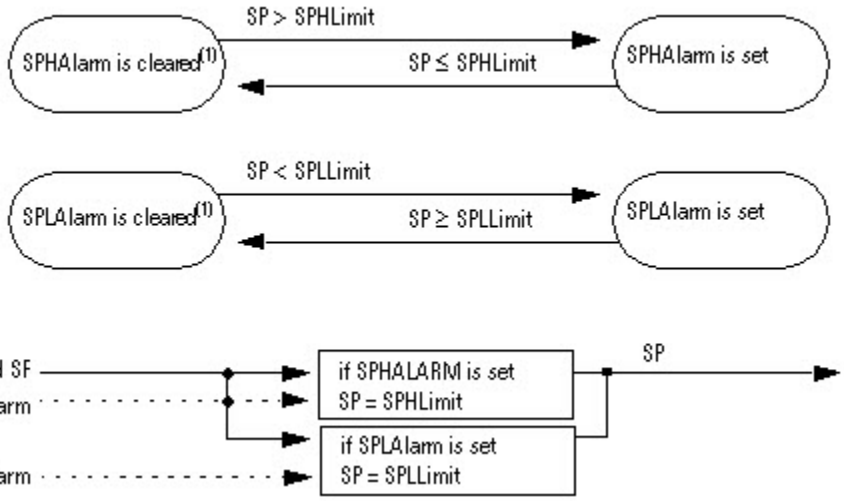
Current SP

The current SP is based on the Cascade/Ratio mode, the PVTracking value, auto mode, and the ProgOper value.



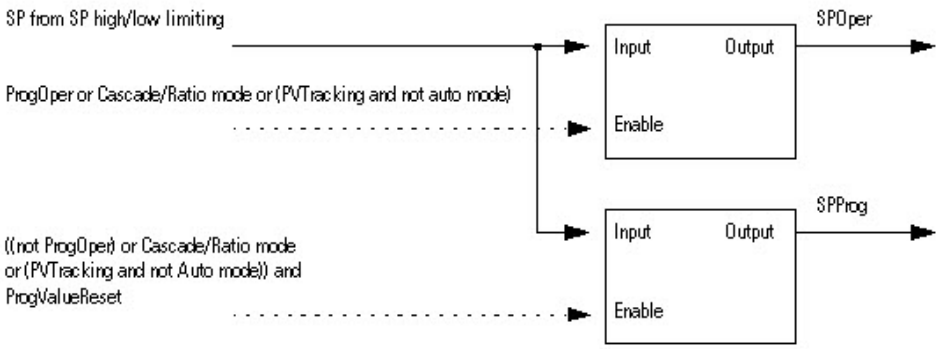
SP High/Low Limiting

The high-to-low alarming algorithm compares SP to the SPHLimit and SPLLimit alarm limits. SPHLimit cannot be greater than PVEUMax and SPLLimit cannot be less than PVEUMin.



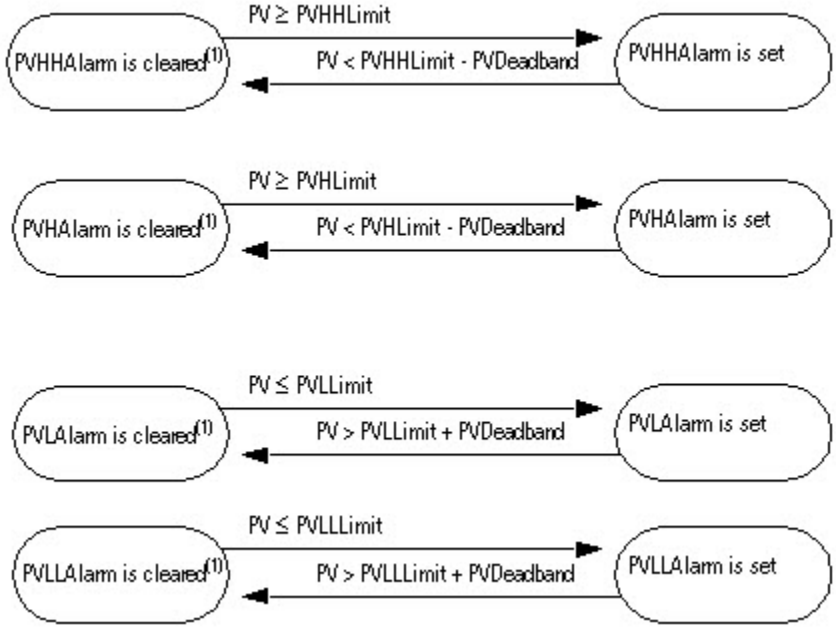
Updating the SPOper and SPProg Values

The PIDE instruction makes SPOper = SP or SPProg = SP to obtain bumpless control switching between Program and Operator control or when switching from Cascade/Ratio mode.



PV High/Low Alarming

The high-high to low-low alarming algorithm compares PV to the PV alarm limits and the PV alarm limits plus or minus the PV alarm deadband



(1) During instruction first scan, the instruction clears all the PV alarm outputs. The instruction also clears the PV alarm outputs and disables the alarming algorithm when PVFaulted is set.

PV Rate-of-Change Alarming

PV rate-of-change (ROC) alarming compares the change in the value of PV over the PVROCPeriod against the PV positive and negative rate-of-change limits. The PVROCPeriod provides a type of deadband for the rate-of-change alarm. For example, if you use a ROC alarm limit of 2°F/second with a period of execution of 100 ms, and an analog input module with a resolution of 1°F, then every time the input value changes, a ROC alarm is generated because the instruction sees a rate of 10°F/second. However, by entering a PVROCPeriod of at least 1 sec, the ROC alarm is only generated if the rate truly exceeds the 2°F/second limit.

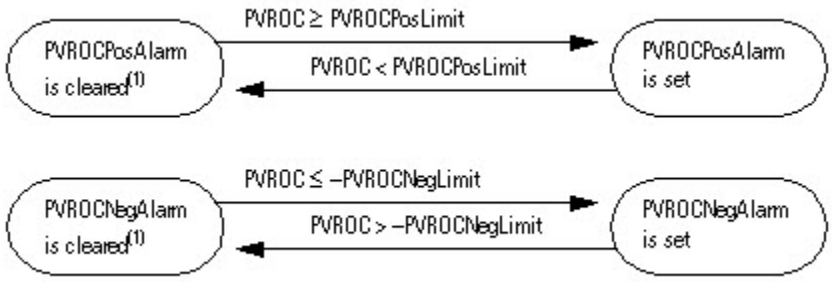
The ROC calculation is only performed when the PVROCPeriod has expired. The rate-of-change is calculated as:

ElapsedROCPeriod = ElapsedROCPeriod + ElapsedTimeSinceLastExecution

If ElapsedROCPeriod ^{3} PVROCPeriod then:

This value:	Is:
PVROC	$\frac{PV_n - PVROC_{n-1}}{PVROCPeriod}$
$PVROC_{N-1}$	$PVROC_{N-1}$ = PV
ElapsedROCPeriod	ElapsedROCPeriod = 0

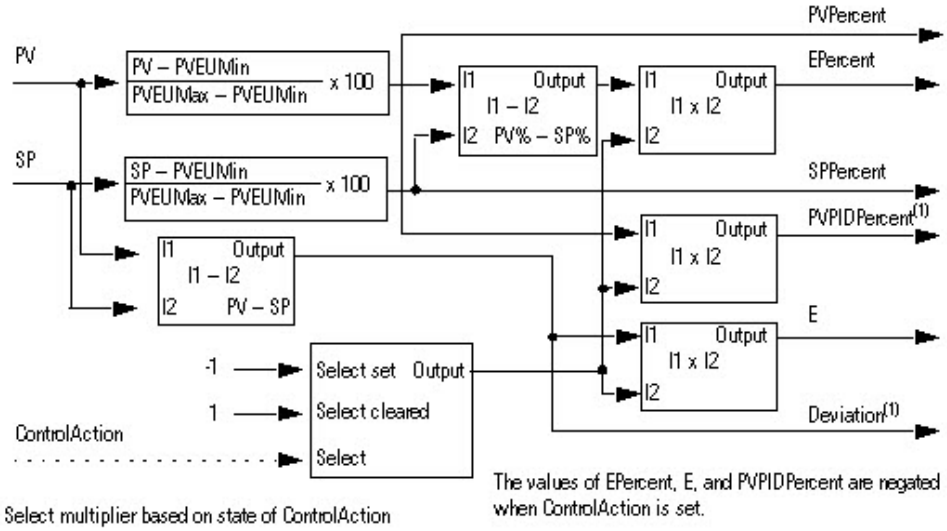
Once PVROC has been calculated, the PV ROC alarms are determined as follows:



(1) During instruction first scan, the instruction clears the PV ROC alarm outputs. The instruction also clears the PVROC alarm outputs and disables the PV ROC alarming algorithm when PVFaulted is set.

Converting the PV and SP Values to Percent

The instruction converts PV and SP to a percent and calculates the error before performing the PID control algorithm. The error is the difference between the PV and SP values. When ControlAction is set, the values of EPercent, E, and PVPIDPercent are negated before being used by the PID algorithm.

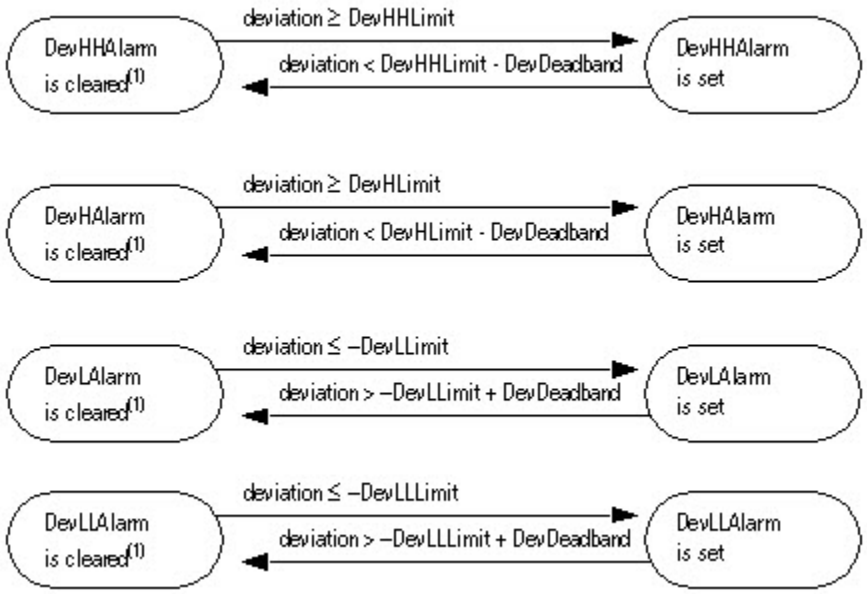


(1) PVPIDPercent and Deviation are internal parameters used by the PID control algorithm.

Deviation High/Low Alarming

Deviation is the difference in value between the process variable (PV) and setpoint (SP). Deviation alarming alerts the operator to a discrepancy between the process variable and the setpoint value.

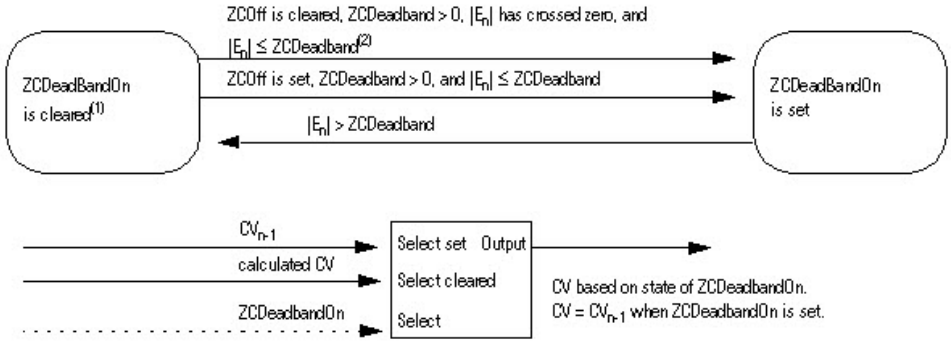
The high-high to low-low alarming algorithm compares the deviation to deviation alarm limits and the deviation alarm limits plus or minus the deadband.



(1) During instruction first scan, the instruction clears the deviation alarm outputs. The instruction also clears the deviation alarm outputs and disables the alarming algorithm when PVFaulted or PVSpanInv is set.

Zero Crossing Deadband Control

You can limit CV such that its value does not change when error remains within the range specified by ZCDeadband (| E | £ ZCDeadband).



(1) When ZCOff is cleared, ZCDeadband > 0, error has crossed zero for the first time, (i.e. En >= 0 and En-1 < 0 or when En <= 0 and En-1 > 0), and | En | <= ZCDeadband, the instruction sets ZCDeadbandOn.

(2) On the transition to Auto or Cascade/Ratio mode, the instruction sets En-1 = En.

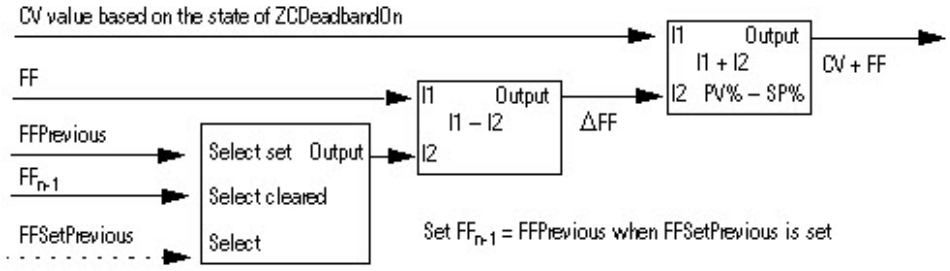
The instruction disables the zero crossing algorithm and clears ZCDeadband under these conditions:

- during instruction first scan
- ZCDeadband £ 0
- Auto or Cascade/Ratio is not the current mode

- PVFaulted is set
- PVSpanInv is set

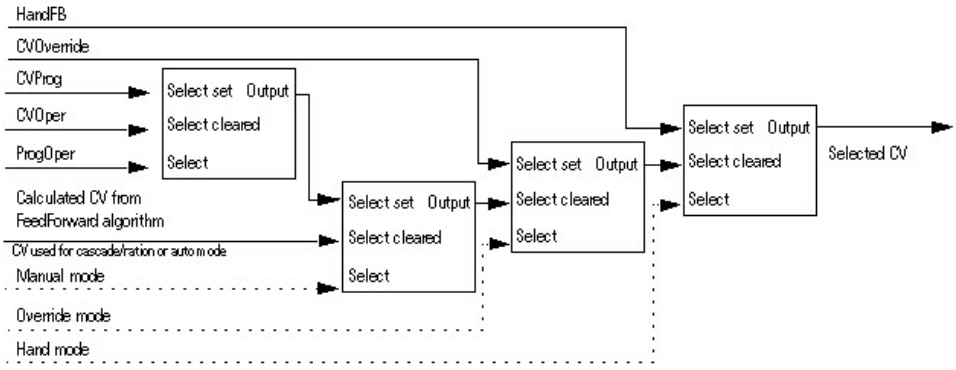
Feedforward Control

Compute CV by summing CV from the zero crossing algorithm with ΔFF. The value of ΔFF = FF – FFn-1. When FFSetPrevious is set, FFn-1=FFPrevious. This lets you preset FFn-1 to a specified value before the instruction calculates the value of ΔFF.



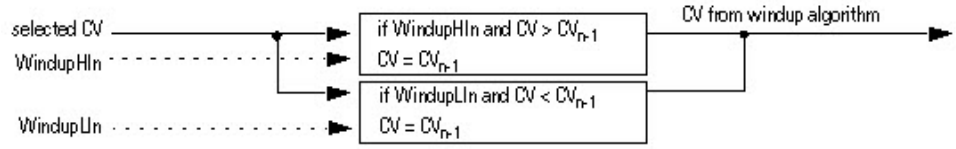
Selecting the Control Variable

Once the PID algorithm has been executed, select the CV based on program or operator control and the current PID mode.



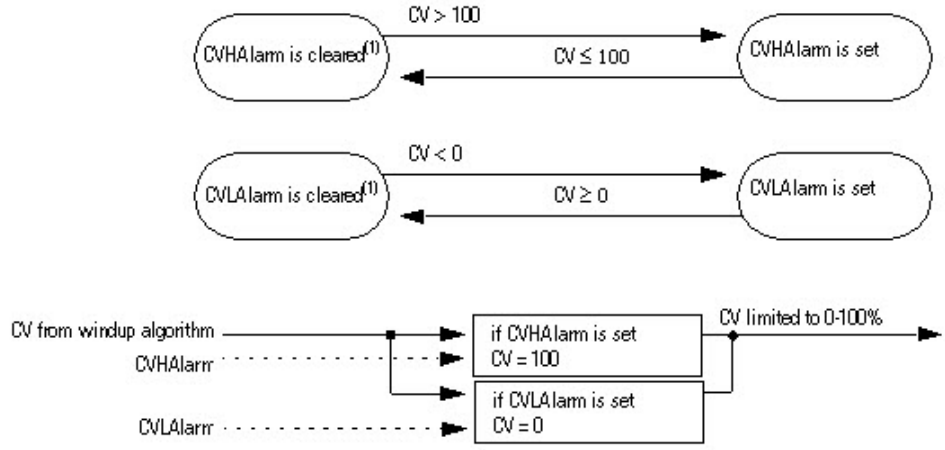
CV Windup Limiting

Limit the CV such that its value cannot increase when WindupHIn is set or decrease when WindupLIn is set. These inputs are typically the WindupHOut or WindupLOut outputs from a secondary loop. The WindupHIn and WindupLIn inputs are ignored if CVInitializing, CVFault, or CVEUSpanInv is set.



CV Percent Limiting

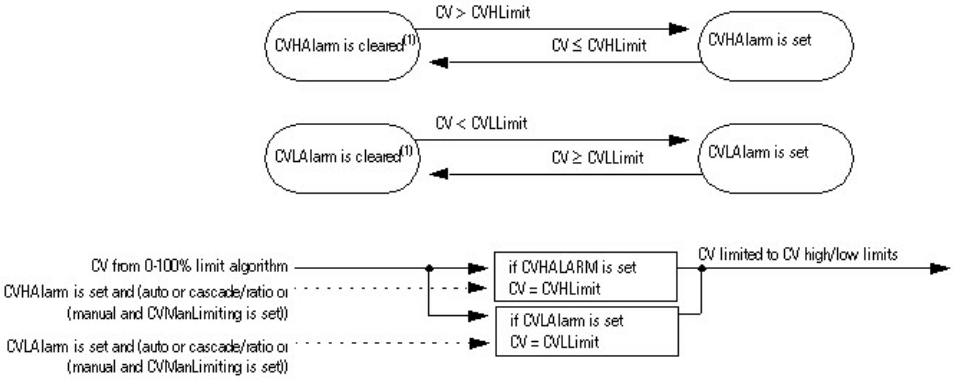
The following diagram illustrates how the instruction determines CV percent limiting.



(1) During instruction first scan, the instruction clears the alarm outputs.

CV High/Low Limiting

The instruction always performs alarming based on CVHLimit and CVLLimit. Limit CV by CVHLimit and CVLLimit when in auto or cascade/ratio mode. When in manual mode, limit CV by CVHLimit and CVLLimit when CVManLimiting is set. Otherwise limit CV by 0 and 100 percent.



(1) During instruction first scan, the instruction clears the alarm outputs.

CV Rate-of-Change Limiting

The PIDE instruction limits the rate-of-change of CV when in Auto or Cascade/Ratio mode or when in Manual mode and CVManLimiting is set. A value of zero disables CV rate-of-change limiting.

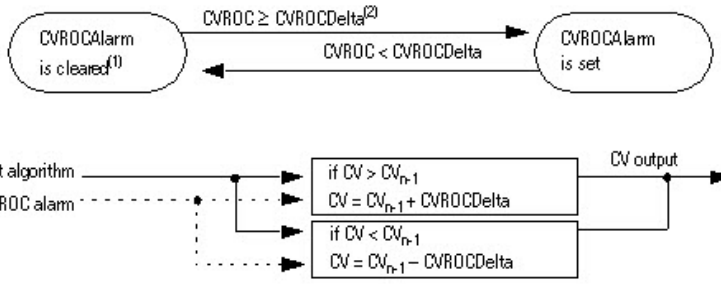
The CV rate-of-change is calculated as:

$$CVROC = |CV_n - CV_{n-1}|$$

$$CVROCDelta = CVROCLimit \times DeltaT$$

where DeltaT is in seconds.

Once CV rate-of-change has been calculated, the CV rate-of-change alarms are determined as follows:

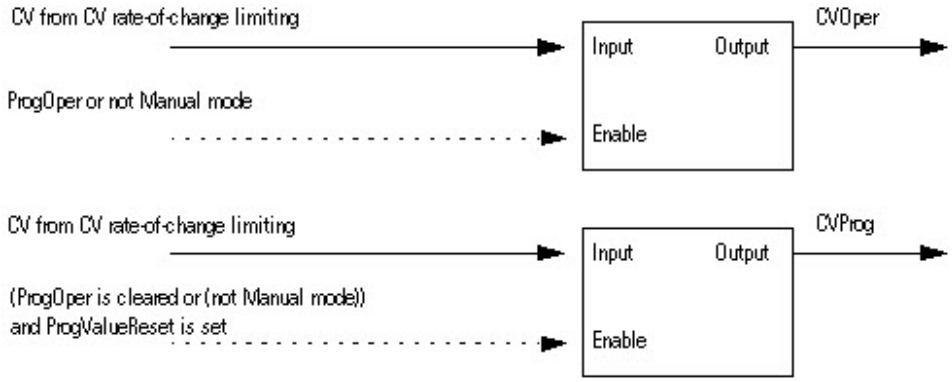


(1) During instruction first scan, the instruction clears the alarm output. The instruction also clears the alarm output and disables the CV rate-of-change algorithm when CVInitializing is set.

(2) When in Auto or Cascade/Ratio mode or when in Manual mode and CVManLimiting is set, the instruction limits the change of CV.

Updating the CVOper and CVProg Values

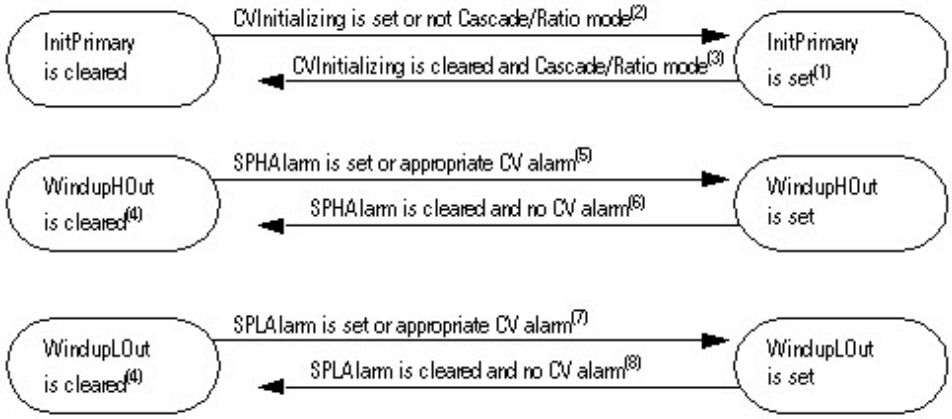
If not in the Operator Manual mode, the PIDE instruction sets CVOper = CV. This obtains bumpless mode switching from any control to the Operator Manual mode.



Primary Loop Control

Primary loop control is typically used by a primary PID loop to obtain bumpless switching and anti-reset windup when using Cascade/Ratio mode. The primary loop control includes the initialize primary loop output and the anti-reset windup outputs. The InitPrimary output is

typically used by the CVInitReq input of a primary PID loop. The windup outputs are typically used by the windup inputs of a primary loop to limit the windup of its CV output.



(1) During instruction first scan, the instruction sets InitPrimary.

(2) When CVInitializing is set or when not in Cascade/Ratio mode the instruction sets InitPrimary.

(3) When CVInitializing is cleared and in Cascade/Ratio mode, the instruction clears InitPrimary.

(4) During instruction first scan, the instruction clears the windup outputs. The instruction also clears the windup outputs and disables the CV windup algorithm when CVInitializing is set or if either CVFaulted or CVEUSpanInv is set.

(5) The instruction sets WindupHOut when SPHAlarm is set, or when ControlAction is cleared and CVHAlarm is set, or when ControlAction is set and CVLAlarm is set.

The SP and CV limits operate independently. A SP high limit does not prevent CV from increasing in value. Likewise, a CV high or low limit does not prevent SP from increasing in value.

(6) The instruction clears WindupHOut when SPHAlarm is cleared, and not (ControlAction is cleared and CVHAlarm is set), and not (ControlAction is set and CVLAlarm is set).

(7) The instruction sets WindupLOut when SPLAlarm is set, or when ControlAction is cleared and CVLAlarm is set, or when ControlAction is set and CVHAlarm is set.

The SP and CV limits operate independently. A SP low limit does not prevent CV from increasing in value. likewise a CV low or high limit does not prevent SP from increasing in value.

(8) The instruction clears WindupLOut when SPLAlarm is cleared and not (ControlAction is cleared and CVLAlarm is set) and not (ControlAction is set and CVHAlarm is set).

Processing Faults

The following table describes how the instruction handles execution faults:

Fault Condition	Action
CVFaulted is true or CVEUSpanInv is true	Instruction is not initialized, CVInitializing is cleared to false Compute PV and SP percent, calculate error, update internal parameters for EPercent and PVPIDPercent PID control algorithm is not executed

Fault Condition	Action
	Disable the Auto and Cascade/Ratio modes. If Override or Hand is not the current mode, set to Manual mode. Set CV to value determined by Program or Operator control and mode (Manual, Override, or Hand).
PVFaulted is true	Disable the Auto and Cascade/Ratio modes. If Override or Hand is not the current mode, set to Manual mode PV high-low, PV rate-of-change, and deviation high-low alarm outputs are cleared to false PID control algorithm is not executed Set CV to value by determined by Program or Operator control and mode (Manual, Override, or Hand).
PVSpanInv is true or SPLimitsInv is true	Disable the Auto and Cascade/Ratio modes. If Override or Hand is not the current mode, set to Manual mode Do not compute PV and SP percent PID control algorithm is not executed Set CV to value by determined by Program or Operator control and mode (Manual, Override, or Hand).
RatioLimitsInv is true and CasRat is true and UseRatio is true	If not already in Hand or Override, set to Manual model Disable the Cascade/Ratio mode Set CV to value determined by Program or Operator control and mode (Manual, Override, or Hand).
TimingModeInv is true or RTSTimeStampInv is true or DeltaTInv is true	If not already in Hand or Override, set to Manual mode

Position Proportional (POSP)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, Compact GuardLogix 5380, CompactLogix 5380, ControlLogix 5580, GuardLogix 5580, and ControlLogix 5590 controllers.

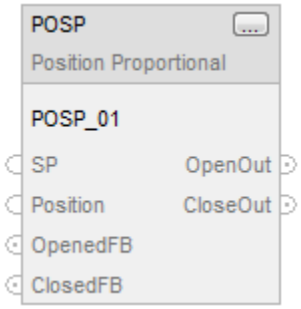
The Position Proportional (POSP) instruction opens or closes a device by pulsing open or close contacts at a user defined cycle time with a pulse width proportional to the difference between the desired and actual positions.

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram logic.

Function Block



Structured Text

POSP(POSP_tag)

Operands

Function Block

Operand	Type	Format	Description
POSP tag	POSITION_PROP	Structure	POSP structure

Structured Text

Operand	Type	Format	Description
block tag	POSITION_PROP	Structure	POSP structure

See *Structured Text Syntax* for more information on the syntax of expressions within structured text.

POSITION_PROP Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If false, the instruction does not execute and outputs are not updated. Default is true.
SP	REAL	Setpoint. This is the desired value for the position. This value must use the same engineering units as Position. Valid = any float Default = 0.0
Position	REAL	Position feedback. This analog input comes from the position feedback from the device. Valid = any float

Input Parameter	Data Type	Description
		Default = 0.0
OpenedFB	BOOL	Opened feedback. This input signals when the device is fully opened. When true, the open output is not allowed to turn on. Default is false.
ClosedFB	BOOL	Closed feedback. This input signals when the device is fully closed. When true, the close output is not allowed to turn on. Default is false.
PositionEUMax	REAL	Maximum scaled value of Position and SP. Valid = any float Default = 100.0
PositionEUMin	REAL	Minimum scaled value of Position and SP. Valid = any float Default = 0.0
CycleTime	REAL	Period of the output pulse in seconds. A value of zero clears both OpenOut and CloseOut. If this value is invalid, the instruction assumes a value of zero and sets the appropriate bit in Status. Valid = any positive float Default = 0.0
OpenRate	REAL	Open rate of the device in %/second. A value of zero clears OpenOut. If this value is invalid, the instruction assumes a value of zero and sets the appropriate bit in Status. Valid = any positive float Default = 0.0
CloseRate	REAL	Close rate of the device in %/second. A value of zero clears CloseOut. If this value is invalid, the instruction assumes a value of zero and sets the appropriate bit in Status. Valid = any positive float Default = 0.0
MaxOnTime	REAL	Maximum time in seconds that an open or close pulse can be on. If OpenTime or CloseTime is calculated

Input Parameter	Data Type	Description
		to be larger than this value, they are limited to this value. If this value is invalid, the instruction assumes a value of CycleTime and sets the appropriate bit in Status. Valid = 0.0 to CycleTime Default = CycleTime
MinOnTime	REAL	Minimum time in seconds that an open or close pulse can be on. If OpenTime or CloseTime is calculated to be less than this value, they are set to zero. If this value is invalid, the instruction assumes a value of zero and sets the appropriate bit in Status. Valid = 0.0 to MaxOnTime Default = 0.0
Deadtime	REAL	Additional pulse time in seconds to overcome friction in the device. Deadtime is added to the OpenTime or CloseTime when the device changes direction or is stopped. If this value is invalid, the instruction sets the appropriate bit in Status and uses a value of Deadtime = 0.0. Valid = 0.0 to MaxOnTime Default = 0.0

Output Parameter	Data Type	Description
EnableOut	BOOL	Indicates if instruction is enabled. Cleared to false if PositionPercent overflows.
OpenOut	BOOL	This output is pulsed to open the device.
CloseOut	BOOL	This output is pulsed to close the device.
PositionPercent	REAL	Position feedback is expressed as percent of the Position span.
SPPercent	REAL	Setpoint is expressed as percent of the Position span.
OpenTime	REAL	Pulse time in seconds of OpenOutput for the current cycle.

Output Parameter	Data Type	Description
CloseTime	REAL	Pulse time in seconds of CloseOutput for the current cycle.
Status	DINT	Status of the function block.
InstructFault (Status.0)	BOOL	The instruction detected one of the following execution errors. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.
CycleTimeInv (Status.1)	BOOL	Invalid CycleTime value. The instruction uses zero.
OpenRateInv (Status.2)	BOOL	Invalid OpenRate value. The instruction uses zero.
CloseRateInv (Status.3)	BOOL	Invalid CloseRate value. The instruction uses zero.
MaxOnTimeInv (Status.4)	BOOL	Invalid MaxOnTime value. The instruction uses the CycleTime value.
MinOnTimeInv (Status.5)	BOOL	Invalid MinOnTime value. The instruction uses zero.
DeadtimeInv (Status.6)	BOOL	Invalid Deadtime value. The instruction uses zero.
PositionPctInv (Status.7)	BOOL	The calculated PositionPercent value is out of range.
SPPercentInv (Status.8)	BOOL	The calculated SPPercent value is out of range.
PositionSpanInv (Status.9)	BOOL	PositionEUMax = PositionEUMin.

Description

The POSP instruction usually receives the desired position setpoint from a PID instruction output.

Scaling the Position and Setpoint Values

The PositionPercent and SPPercent outputs are updated each time the instruction is executed. If either of these values is out of range (less than 0% or greater than 100%), the appropriate bit

in Status is set, but the values are not limited. The instruction uses these formulas to calculate whether the values are in range:

$$PositionPercent = \frac{Position - PositionEUMin}{PositionEUMax - PositionEUMin} \times 100$$

$$SPPercent = \frac{SP - PositionEUMin}{PositionEUMax - PositionEUMin} \times 100$$

How the POSP Instruction Uses the Internal Cycle Timer

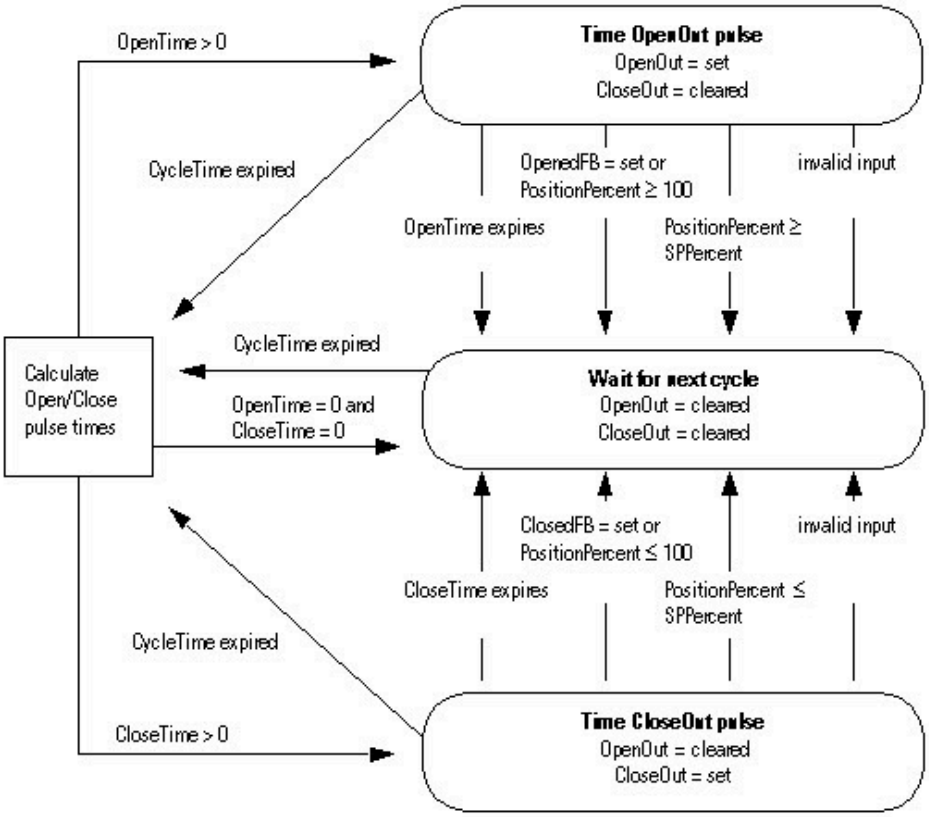
The instruction uses CycleTime to determine how often to recalculate the duration of Open and Close output pulses. An internal timer is maintained and updated by DeltaT. DeltaT is the elapsed time since the instruction last executed. Whenever the internal timer equals or exceeds the programmed CycleTime (cycle time expires) the Open and Close outputs are recalculated.

You can change the CycleTime at any time.

If CycleTime = 0, the internal timer is cleared to 0, OpenOut is cleared to false and CloseOut is cleared to false.

Producing Output Pulses

The following diagram shows the three primary states of the POSP instruction.



Calculating Open and Close Pulse Times

OpenOut is pulsed whenever SP > Position feedback. When this occurs, the instruction sets CloseTime = 0 and the duration for which OpenOut is to be turned on is calculated as:

$$OpenTime = \frac{SPPercent - PositionPercent}{OpenRate}$$

If OpenTimen-1 < CycleTime, then add Deadtime to OpenTime.

If OpenTime > MaxOnTime, then limit to MaxOnTime.

If OpenTime < MinOnTime, then set OpenTime = 0.

If any of the following conditions exist, OpenOut is not pulsed and OpenTime = 0.

OpenFB is true or PositionPercent ≥ 100

CycleTime = 0

OpenRate = 0

SPPercent is invalid

The CloseOut is pulsed whenever SP < Position feedback. When this occurs, the instruction sets OpenTime = 0 and the duration for which CloseOut is to be turned on is calculated as:

$$CloseTime = \frac{PositionPercent - SPPercent}{CloseRate}$$

If CloseTimen-1 < CycleTime, then add Deadtime to CloseTime.

If CloseTime > MaxOnTime, then limit to MaxOnTime.

If CloseTime < MinOnTime, then set CloseTime to 0.

If any of the following conditions exist, CloseOut will not be pulsed and CloseTime will be cleared to 0.0.

ClosedFB is true or PositionPercent ≤ 0

CycleTime = 0

CloseRate = 0

SPPercent is invalid

OpenOut and CloseOut will not be pulsed if SPPercent equals PositionPercent. Both OpenTime and CloseTime will be cleared to false.

Affects Math Status Flags

No

Major/Minor Faults

None specific to this instruction. See Common Attributes on page for operand-related faults.

Execution

Function Block

Condition/State	Action Taken
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is true	EnableIn and EnableOut bits are set to true. The instruction executes.
Instruction first run	N/A
Instruction first scan	OpenTime and CloseTime are cleared to 0.0.
Postscan	EnableIn and EnableOut bits are cleared to false.

Structured Text

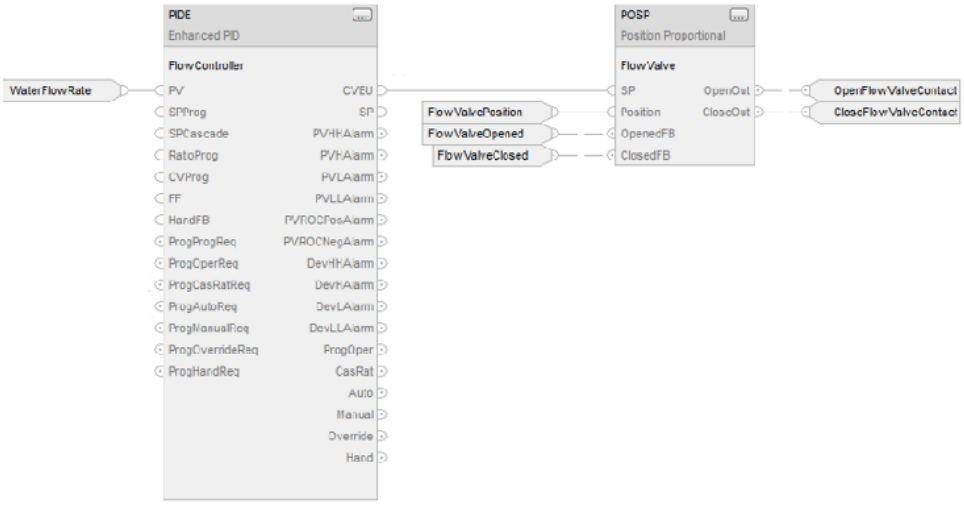
Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.
Normal Execution	See Tag.EnableIn is true in the Function Block table.
Postscan	See Postscan in the Function Block table.

Examples

Example 1

In this example, the POSP instruction opens or closes a motor-operated valve based on the CVEU output of the PIDE instruction. The actual valve position is wired into the Position input and optional limit switches, which show if the valve is fully opened or closed, are wired into the OpenedFB and ClosedFB inputs. The OpenOut and CloseOut outputs are wired to the open and close contacts on the motor-operated valve.

Function Block



Structured Text

```
FlowController.PV := WaterFlowRate;
PIDE(FlowController);

FlowValve.SP := FlowController.CVEU;
FlowValve.Position := FlowValvePosition;
FlowValve.OpenedFB := FlowValveOpened;
FlowValve.ClosedFB := FlowValveClosed;
POSP(FlowValve);

OpenFlowValveContact := FlowValve.OpenOut;
CloseFlowValveContact := FlowValve.CloseOut;
```

Ramp/Soak (RMPS)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, Compact GuardLogix 5380, CompactLogix 5380, ControlLogix 5580, GuardLogix 5580, and ControlLogix 5590 controllers.

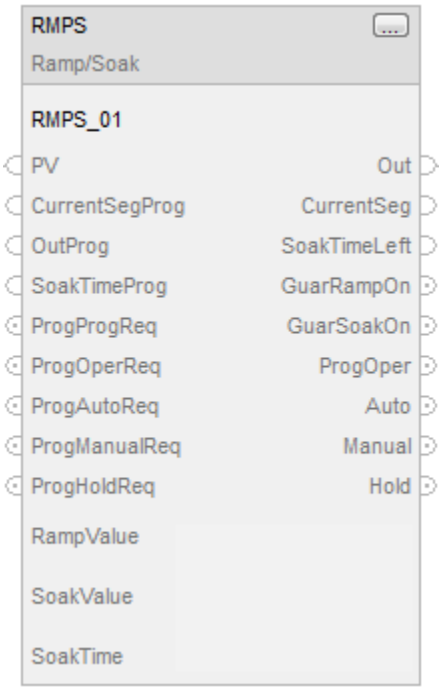
The Ramp/Soak (RMPS) instruction provides for a number of segments of alternating ramp and soak periods.

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram logic.

Function Block



Structured Text

RMPS(RMPS_tag,RampValue,SoakValue,SoakTime);

Operands

Function Block

Operand	Type	Format	Description
RMPS tag	RAMP_SOAK	structure	RMPS structure
RampValue	REAL	array	Ramp Value array. Enter a ramp value for each segment (0 to NumberOfSegs-1). Ramp values are entered as time in minutes or as a rate in units/minute. The TimeRate parameter reflects which method is used to specify the ramp. If a ramp value is invalid, the instruction sets the appropriate bit in Status and changes to Operator Manual or Program Hold mode. The array must be at least as large as NumberOfSegs.

Operand	Type	Format	Description
			Valid = 0.0 to maximum positive float
SoakValue	REAL	array	Soak Value array. Enter a soak value for each segment (0 to NumberOfSegs-1). The array must be at least as large as NumberOfSegs. Valid = any float
SoakTime	REAL	array	Soak Time array. Enter a soak time for each segment (0 to NumberOfSegs-1). Soak times are entered in minutes. If a soak value is invalid, the instruction sets the appropriate bit in Status and changes to Operator Manual or Program Hold mode. The array must be at least as large as NumberOfSegs. Valid = 0.0 to maximum positive float

Structured Text

Operand	Type	Format	Description
RMPS tag	RAMP_SOAK	structure	RMPS structure
RampValue	REAL	array	Ramp Value array. Enter a ramp value for each segment (0 to NumberOfSegs-1). Ramp values are entered as time in minutes or as a rate in units/minute. The TimeRate parameter reflects which method is used to specify the ramp. If a ramp value is invalid, the instruction sets the appropriate bit in Status and changes to Operator Manual or Program Hold mode. The array must be at least as large as NumberOfSegs.

Operand	Type	Format	Description
			Valid = 0.0 to maximum positive float
SoakValue	REAL	array	Soak Value array. Enter a soak value for each segment (0 to NumberOfSegs-1). The array must be at least as large as NumberOfSegs. Valid = any float
SoakTime	REAL	array	Soak Time array. Enter a soak time for each segment (0 to NumberOfSegs-1). Soak times are entered in minutes. If a soak value is invalid, the instruction sets the appropriate bit in Status and changes to Operator Manual or Program Hold mode. The array must be at least as large as NumberOfSegs. Valid = 0.0 to maximum positive float

See *Structured Text Syntax* for more information on the syntax of expressions within structured text.

RMPS Structure

Specify a unique RMPS structure for each instruction.

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If false, the instruction does not execute and outputs are not updated. Default is true.
PV	REAL	The scaled analog temperature signal input to the instruction. Valid = any float Default = 0.0
PVFault	BOOL	Bad health indicator of PV. If true, the input is invalid, the instruction is placed in Program Hold or Operator Manual mode, and the instruction sets the appropriate bit in Status. Default is false.

Input Parameter	Data Type	Description
NumberOfSegs	DINT	Number of segments. Specify the number of ramp/soak segments used by the instruction. The arrays for RampValue, SoakValue, and SoakTime must be at least as large as NumberOfSegs. If this value is invalid, the instruction is placed into Operator Manual or Program Hold mode and the instruction sets the appropriate bit in Status. Valid = 1 to (minimum size of RampValue, SoakValue, or SoakTime arrays) Default = 1
ManHoldAftInit	BOOL	Manual/Hold after initialization. If true, the ramp/soak is in Operator Manual or Program Hold mode after initialization completes. Otherwise, the ramp/soak remains is in its previous mode after initialization completes. Default is false.
CyclicSingle	BOOL	Cyclic/single execution. True for cyclic action or false for single action. Cyclic action continuously repeats the ramp/soak profile. Single action performs the ramp/ soak profile once and then stops. Default is false.
TimeRate	BOOL	Time/rate ramp value configuration. True if the RampValue parameters are entered as a time in minutes to reach the soak temperature. False if the RampValue parameters are entered as a rate in units/minute. Default is false.
GuarRamp	BOOL	Guaranteed ramp. If true and the instruction is in Auto, ramping is temporarily suspended if the PV differs from the Output by more than RampDeadband. Default is cleared.
RampDeadband	REAL	Guaranteed ramp deadband. Specify the amount in engineering units that PV is allowed to differ from the output when GuarRamp is on. If this value is invalid, the instruction sets RampDeadband = 0.0 and the

Input Parameter	Data Type	Description
		instruction sets the appropriate bit in Status. Valid = any float ≥ 0.0 Default = 0.0
GuarSoak	BOOL	Guaranteed soak. If true and the instruction is in auto, the soak timer is cleared if the PV differs from the Output by more than SoakDeadband. Default is false.
SoakDeadband	REAL	Guaranteed soak deadband. Specify the amount in engineering units that the PV is allowed to differ from the output when GuarSoak is on. If this value is invalid, the instruction sets SoakDeadband = 0.0 and the instruction sets the appropriate bit in Status. Valid = any float ≥ 0.0 Default = 0.0
CurrentSegProg	DINT	Current segment program. The user program writes a requested value for the CurrentSeg into this input. This value is used if the ramp/soak is in Program Manual mode. If this value is invalid, the instruction sets the appropriate bit in Status. Valid = 0 to NumberOfSegs-1 Default = 0
OutProg	REAL	Output program. The user program writes a requested value for the Out into this input. This value is used as the Out when the ramp/soak is in Program Manual mode. Valid = any float Default = 0.0
SoakTimeProg	REAL	Soak time program. The user program writes a requested value for the SoakTimeLeft into this input. This value is used if the ramp/soak is in Program Manual mode. If this value is invalid, the instruction sets the appropriate bit in Status. Valid = 0.0 to maximum positive float Default = 0.0
CurrentSegOper	DINT	Current segment operator. The operator interface writes a

Input Parameter	Data Type	Description
		requested value for the CurrentSeg into this input. This value is used if the ramp/soak is in Operator Manual mode. If this value is invalid, the instruction sets the appropriate bit in Status. Valid = 0 to NumberOfSegs-1 Default = 0
OutOper	REAL	Output operator. The operator interface writes a requested value for the Out into this input. This value is used as the Out when the ramp/soak is in Operator Manual mode. Valid = any float Default = 0.0
SoakTimeOper	REAL	Soak time operator. The operator interface writes a requested value for the SoakTimeLeft into this input. This value is used if the ramp/soak is in Operator Manual mode. If this value is invalid, the instruction sets the appropriate bit in Status. Valid = 0.0 to maximum positive float Default = 0.0
ProgProgReq	BOOL	Program program request. Set to true by the user program to request Program control. Ignored if ProgOperReq is true. Holding this true and ProgOperReq false locks the instruction in Program control. Default is false.
ProgOperReq	BOOL	Program operator request. Set to true by the user program to request Operator control. Holding this true locks the instruction in Operator control. Default is false.
ProgAutoReq	BOOL	Program auto mode request. Set to true by the user program to request the ramp/soak to enter Auto mode. Ignored if the loop is in Operator control, if ProgManualReq is true, or if ProgHoldReq is true. Default is false.
ProgManualReq	BOOL	Program manual mode request. Set to true by the user program

Input Parameter	Data Type	Description
		to request the ramp/soak to enter Manual mode. Ignored if the ramp/soak is in Operator control or if ProgHoldReq is true. Default is false.
ProgHoldReq	BOOL	Program hold mode request. Set to true by the user program to request to stop the ramp/ soak without changing the Out, CurrentSeg, or SoakTimeLeft. Also useful when a PID loop getting its setpoint from the ramp/soak leaves cascade. An operator can accomplish the same thing by placing the ramp/soak into Operator Manual mode. Default is false.
OperProgReq	BOOL	Operator program request. Set to true by the operator interface to request Program control. Ignored if ProgOperReq is true. The instruction clears this input to false. Default is false.
OperOperReq	BOOL	Operator operator request. Set to true by the operator interface to request Operator control. Ignored if ProgProgReq is true and ProgOperReq is false. The instruction clears this input to false. Default is false.
OperAutoReq	BOOL	Operator auto mode request. Set to true by the operator interface to request the ramp/soak to enter Auto mode. Ignored if the loop is in Program control or if OperManualReq is true. The instruction clears this input to false. Default is false.
OperManualReq	BOOL	Operator manual mode request. Set to true by the operator interface to request the ramp/soak to enter Manual mode. Ignored if the loop is in Program control. The instruction clears this input to false. Default is false.
Initialize	BOOL	Initialize program and operator values. When true and in manual, the instruction sets CurrentSegProg = 0, CurrentSegOper = 0, SoakTimeProg

Input Parameter	Data Type	Description
		= SoakTime[0], and SoakTimeOper = SoakTime[0]. Initialize is ignored when in Auto or Hold mode. The instruction clears this parameter to false. Default is false.
ProgValueReset	BOOL	Reset program control values. When true, the instruction clears ProgProgReq, ProgOperReq, ProgAutoReq, ProgHoldReq, and ProgManualReq to false. Default is false.

Output Parameter	Data Type	Description
EnableOut	BOOL	Indicates if instruction is enabled. Cleared to false if Out overflows.
Out	REAL	The output of the ramp/soak instruction.
CurrentSeg	DINT	Current segment number. Displays the current segment number in the ramp/soak cycle. Segments start numbering at 0.
SoakTimeLeft	REAL	Soak time left. Displays the soak time remaining for the current soak.
GuarRampOn	BOOL	Guaranteed ramp status. Set to true if the Guaranteed Ramp feature is in use and the ramp is temporarily suspended because the PV differs from the output by more than the RampDeadband.
GuarSoakOn	BOOL	Guaranteed soak status. Set to true if the Guaranteed Soak feature is in use and the soak timer is cleared because the PV differs from the output by more than the SoakDeadband.
ProgOper	BOOL	Program/Operator control indicator. True when in Program control. False when in Operator control.

Output Parameter	Data Type	Description
Auto	BOOL	Auto mode. True when the ramp/soak is in Program Auto or Operator Auto mode.
Manual	BOOL	Manual mode. True when the ramp/soak is in Program Manual or Operator Manual mode.
Hold	BOOL	Hold mode. True when the ramp/soak is in program Hold mode.
Status	DINT	Status of the function block.
InstructFault (Status.0)	BOOL	The instruction detected one of the following execution errors. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.
PVFaulted (Status.1)	BOOL	PVHealth is bad.
NumberOfSegsInv (Status.2)	BOOL	The NumberOfSegs value is invalid value or is not compatible with an array size.
RampDeadbandInv (Status.3)	BOOL	Invalid RampDeadband value.
SoakDeadbandInv (Status.4)	BOOL	Invalid SoakDeadband value.
CurrSegProgInv (Status.5)	BOOL	Invalid CurrSegProg value.
SoakTimeProgInv (Status.6)	BOOL	Invalid SoakTimeProg value.
CurrSegOperInv (Status.7)	BOOL	Invalid CurrSegOper value.
SoakTimeOperInv (Status.8)	BOOL	Invalid SoakTimeOper value.
RampValueInv (Status.9)	BOOL	Invalid RampValue value.
SoakTimeInv (Status.10)	BOOL	Invalid SoakTime value

Description

The RMPS instruction is typically used to provide a temperature profile in a batch heating process. The output of this instruction is typically the input to the setpoint of a PID loop.

Whenever the value computed for the output is invalid, NAN, or ± INF, the instruction sets Out = the invalid value and sets the math overflow status flag. The internal parameters are not updated. In each subsequent scan, the output is computed using the internal parameters from the last scan when the output was valid.

Monitoring the RMPS Instruction

There is an operator faceplate available for the RMPS instruction.

Affects Math Status Flags

No

Major/Minor Faults

None specific to this instruction. See Common Attributes on page for operand-related faults.

Execution

Function Block

Condition/State	Action Taken
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is true	EnableIn and EnableOut bits are set to true. The instruction executes.
Instruction first run	Clear CurrentSeg to 0. Mode is set to operator manual mode. SoakTimeProg and SoakTimeOper are set to SoakTime[0] if SoakTime[0] is valid.
Instruction first scan	All the operator request inputs are cleared to false. If ProgValueReset is true, all the program request inputs are cleared to false. The operator control mode is set to manual mode if the current mode is hold.
Postscan	EnableIn and EnableOut bits are cleared to false.

Structured Text

Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.
Normal Execution	See Tag.EnableIn is true in the Function Block table.
Postscan	See Postscan in the Function Block table.

Initial Mode Applied on Instruction First Scan

The following table shows the ending control based on the program request inputs.

Control at Start of First Scan	ProgOperReq	ProgProgReq	ProgValueReset	FirstRun	Control at End of First Scan
Operator Control	false	true	false	na	Program Control
	na	false	na	na	Operator Control
Program Control	true	Na	false	false	Operator Control
	na	Na	true	true	Operator Control
	false	false	false	true	Operator Control
	false	true	false	na	Program Control
	na	Na	true	false	Program Control
	false	false	false	false	Program Control

The following table shows the ending control based on the Manual, Auto, and Hold mode requests.

Control at Start of First Scan	Oper Auto Req	Oper Man Req	Prog Auto Req	Prog Man Req	Prog Hold Req	Man Hold AftInit	Prog Value Reset	First Run	Control at End of First Scan
Operator Control	na	na	na	na	na	false	na	false	Operator Current mode
	na	na	na	na	na	na	na		Operator Manual mode
	na	na	na	na	na	true	na	na	
Program Control	na	na	false	false	false	false	na	false	Program Current mode
	na	na	na	na	na	false	true	false	
	na	na	true	false	false	false	false	na	Program Auto mode
	na	na	na	true	false	false	false	na	Program Manual mode
	na	na	na	na	true	false	false	na	Program Hold mode

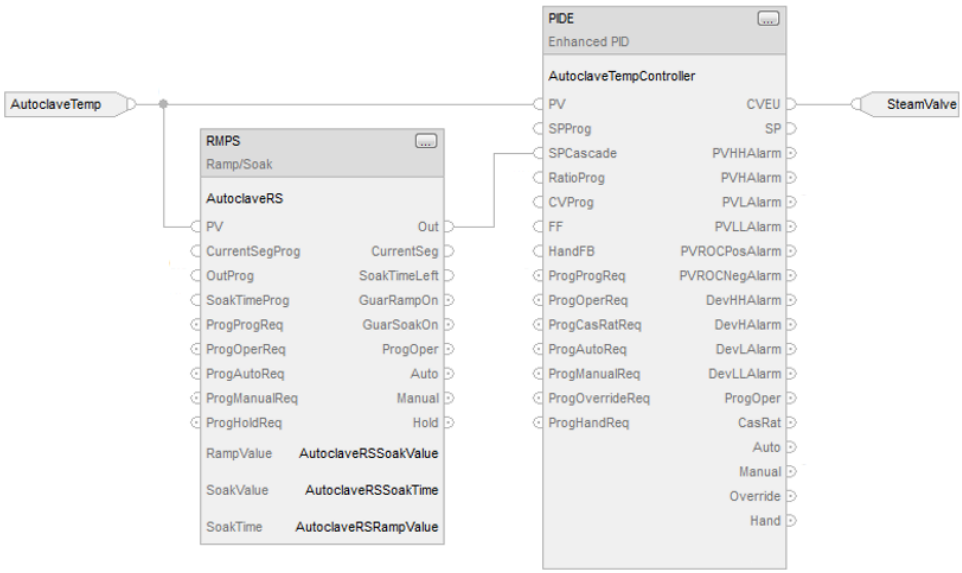
Control at Start of First Scan	Oper Auto Req	Oper Man Req	Prog Auto Req	Prog Man Req	Prog Hold Req	Man Hold AftInit	Prog Value Reset	First Run	Control at End of First Scan
	na	na	na	na	na	true	na	na	

Example

In this example, the RMPS instruction drives the setpoint of a PIDE instruction. When the PIDE instruction is in Cascade/Ratio mode, the output of the RMPS instruction is used as the setpoint. The PV to the PIDE instruction can be optionally fed into the PV input of the RMPS instruction if you want to use guaranteed ramping and/or guaranteed soaking.

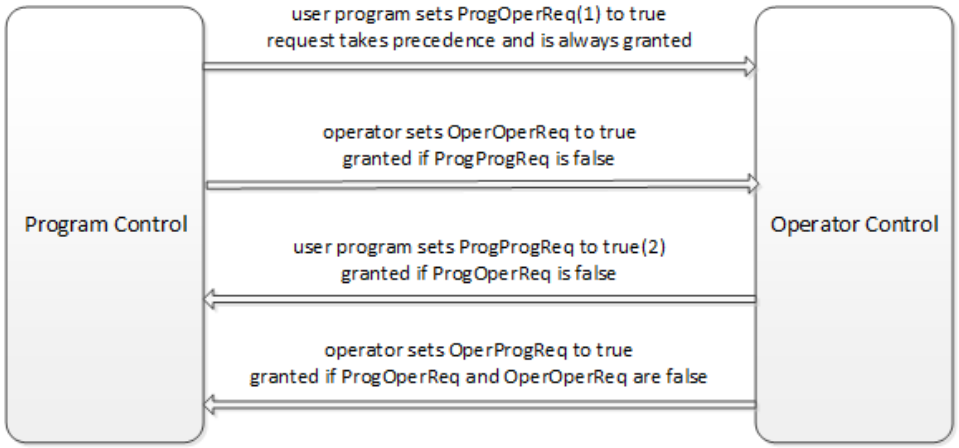
In this example, the AutoclaveRSSoakValue, AutoclaveRSSoakTime, and AutoclaveRSRampValue arrays are REAL arrays with 10 elements to allow up to a 10 segment RMPS profile.

Function Block



Switching Between Program Control and Operator Control

The RMPS instruction can be controlled by either a user program or through an operator interface. Control can be changed at any time.



(1) You can lock the instruction in Operator control by leaving ProgOperReq to true.

(2) You can lock the instruction in Program control by leaving ProgProgReq to true while ProgOperReq is false.

When transitioning from Operator control to Program control while the ProgAutoReq, ProgManualReq, and ProgHoldReq inputs are false, the mode is determined as follows:

If the instruction was in Operator Auto mode, then the transition is to Program Auto mode.

If the instruction was in Operator Manual mode, then the transition is to Program Manual mode.

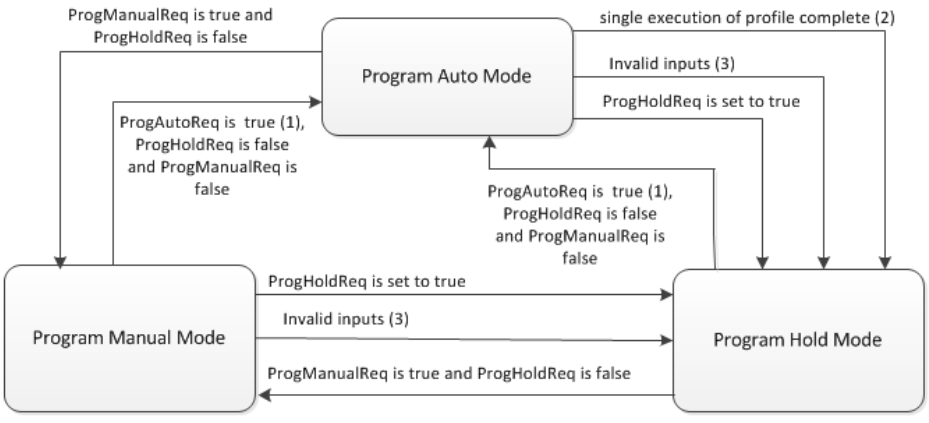
When transitioning from Program control to Operator control while the OperAutoReq and OperManualReq inputs are false, the mode is determined as follows:

If the instruction was in Program Auto mode, then the transition is to Operator Auto mode.

If the instruction was in Program Manual or Program Hold mode, then the transition is to Operator Manual mode.

Program Control

The following diagram shows how the RMPS instruction operates when in Program control.



(1) In single (non-cyclic) execution, you must toggle ProgAutoReq from false to true if one execution of the ramp/soak profile is complete and you want another execution of the ramp/soak profile.

(2) When the instruction is configured for single execution, and the Auto mode Ramp-Soak profile completes, the instruction transitions to Hold mode.

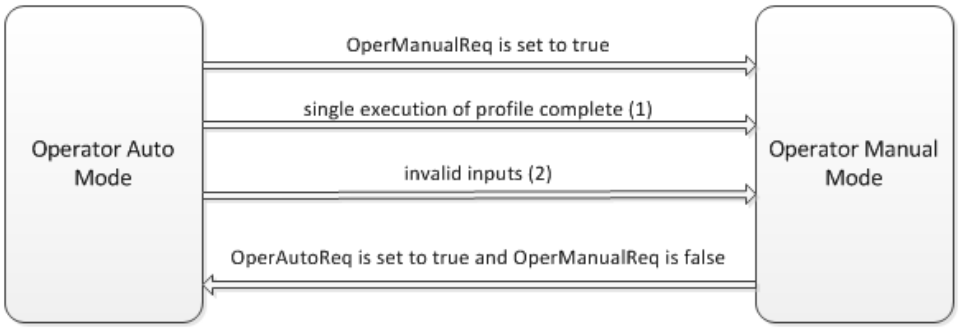
(3) The instruction is placed in Hold mode if PVFaulted is true or any of the following inputs are invalid: NumberOfSegs, CurrentSegProg, or SoakTimeProg.

The following table describes the possible Program modes.

Mode	Description
Program Auto Mode	While in Auto mode, the instruction sequentially executes the ramp/ soak profile.
Program Manual Mode	While in Manual mode the user program directly controls the instruction's Out. The CurrentSegProg, SoakTimeProg, and OutProg inputs are transferred to the CurrentSeg, SoakTimeLeft, and Out outputs. When the instruction is placed in auto mode, the ramp/soak function resumes with the values last input from the user program. CurrentSegProg and SoakTimeProg are not transferred if they are invalid. To facilitate a "bumpless" transition into Manual mode, the CurrentSegProg, SoakTimeProg, and OutProg inputs are continuously updated to the current values of CurrentSeg, SoakTimeLeft, and Out when ProgValueReset is set and the instruction is not in Program Manual mode.
Program Hold Mode	While in Hold mode, the instruction's outputs are maintained at their current values. If in this mode when ProgOperReq is set to change to Operator control, the instruction changes to Operator Manual mode.

Operator Control

The following diagram shows how the RMPS instruction operates when in Operator control.



(1) When the instruction is configured for Single Execution, and the Auto mode ramp/soak profile completes, the instruction transitions to manual mode.

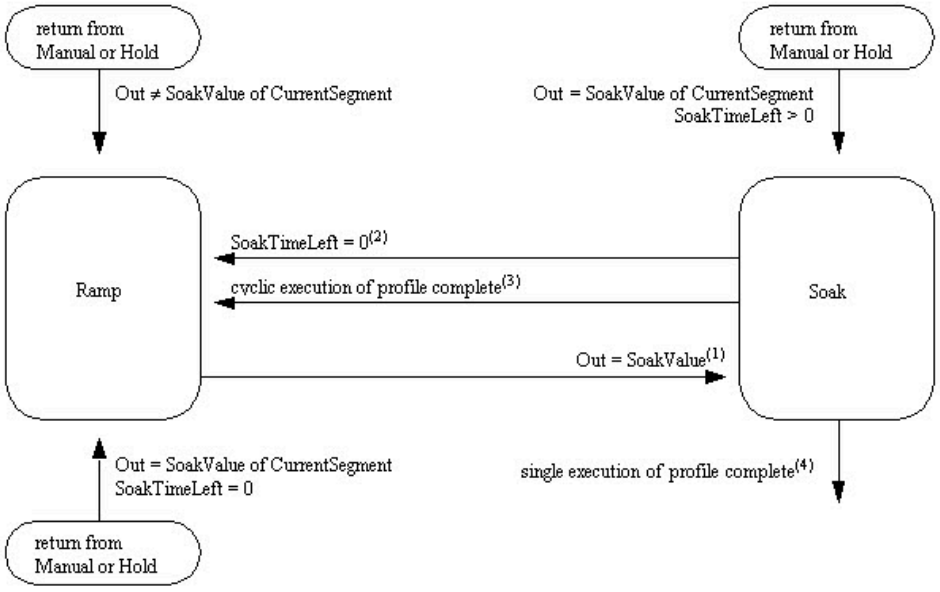
(2) The instruction is placed in Manual mode if PVFaulted is true or any of the following inputs are invalid: NumberOfSegs, CurrentSegOper, or SoakTimeOper.

The following table describes the possible Operator modes.

Mode	Description
Operator Auto Mode	While in Auto mode, the instruction sequentially executes the ramp/ soak profile.
Operator Manual Mode	While in Manual mode the operator directly controls the instruction's Out. The CurrentSegOper, SoakTimeOper, and OutOper inputs are transferred to the CurrentSeg, SoakTimeLeft, and Out outputs. When the instruction is placed in Auto mode, the ramp/soak function resumes with the values last input from the operator. CurrentSegOper and SoakTimeOper are not transferred if they are invalid. To facilitate a "bumpless" transition into Manual mode, the CurrentSegOper, SoakTimeOper, and OutOper inputs are continuously updated to the current values of CurrentSeg, SoakTimeLeft, and Out whenever the instruction is not in Operator Manual mode.

Executing the Ramp/Soak Profile

The following diagram illustrates how the RMPS instruction executes the ramp/soak profile.



(1) The Ramp is complete when Out = SoakValue. If, during ramp execution, Out > SoakValue, Out is limited to SoakValue.

(2) Soaking is complete when Out is held for the amount of time specified in the current segment's SoakTime. If the segment executed was not the last segment, CurrentSeg increments by one.

(3) Soaking has completed for the last programmed segment and the instruction is configured for cyclic execution. The instruction sets CurrentSeg = 0.

(4) Soaking has completed for the last programmed segment and the instruction is configured for single execution.

(5) When returning to Auto mode, the instruction determines if ramping or soaking resumes. What to do next depends on the values of Out, SoakTimeLeft, and the SoakValue of the current segment. If Out = SoakValue for the current segment, and SoakTimeLeft = 0, then the current segment has completed and the next segment starts.

Ramping

The ramp cycle ramps Out from the previous segment's SoakValue to the current segment's SoakValue. The time in which the ramp is traversed is defined by the RampValue parameters.

Ramping is positive if Out < target SoakValue of the current segment. If the ramp equation calculates a new Out which exceeds the target SoakValue, the Out is set to the target SoakValue.

Ramping is negative if Out > the target SoakValue of the current segment. If the ramp equation calculates a new Out which is less than the target SoakValue, the Out is set to the target SoakValue.

Each segment has a ramp value. You have the option of programming the ramp in units of time or rate. All segments must be programmed in the same units. The following table describes the ramping options:

Parameter	Description
Time-based ramping	TimeRate is set to true for time-based ramping (in minutes) The rate of change for the current segment is calculated and either added or subtracted to Out until Out reaches the current segment's soak value. In the following equation, DeltaT is the elapsed time in minutes since the instruction last executed. $Out = Out \pm \frac{(SoakValue_{CurrentSeg} - RampStart)}{RampValue_{CurrentSeg}} \times \Delta t$ Where RampStart is the value of Out at the start of the Current Segment.
Rate-based ramping	TimeRate is cleared to false for rate-based ramping (in units/minute) The programmed rate of change is either added or subtracted to Out until Out reaches the current segment's soak value. In the following equation, DeltaT is the elapsed time in minutes since the instruction last executed. $Out = Out \pm RampValue_{CurrentSeg} \times \Delta t$

Guaranteed Ramping

Set the input GuarRamp to true to enable guaranteed ramping. When enabled, the instruction monitors the difference between Out and PV. If the difference is outside of the programmed RampDeadband, the output is left unchanged until the difference between PV and Out are within the deadband. The output GuarRampOn is set to true whenever Out is held due to guaranteed ramping being in effect.

Soaking

Soaking is the amount of time the block output is to remain unchanged until the next ramp-soak segment is started. The soak cycle holds the output at the SoakValue for a programmed amount of time before proceeding to the next segment. The amount of time the output is to soak is programmed in the SoakTime parameters.

Each segment has a SoakValue and SoakTime. Soaking begins when Out is ramped to the current segment's SoakValue. SoakTimeLeft represents the time in minutes remaining for the output to soak. During ramping, SoakTimeLeft is set to the current segment's SoakTime. Once ramping is complete, SoakTimeLeft is decreased to reflect the time in minutes remaining for the current segment. SoakTimeLeft = 0 when SoakTime expires.

Guaranteed Soaking

Set the input GuarSoak to true to enable guaranteed soaking. When enabled, the instruction monitors the difference between Out and PV. If the difference is outside of the SoakDeadband, timing of the soak cycle is suspended and the internal soak timer is cleared. When the difference between Out and PV returns to within the deadband, timing resumes. The output GuarSoakOn is set to true when timing is held due to guaranteed soaking being in effect.

Scale (SCL)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, Compact GuardLogix 5380, CompactLogix 5380, ControlLogix 5580, GuardLogix 5580, and ControlLogix 5590 controllers.

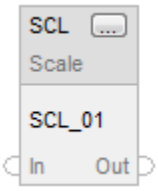
The Scale (SCL) instruction converts an unscaled input value to a floating point value in engineering units.

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram logic.

Function Block



Structured Text

SCL(SCL_tag)

Operands

Function Block

Operand	Type	Format	Description

SCL tag	SCALE	Structure	SCL structure

Structured Text

Operand	Type	Format	Description
SCL tag	SCALE	Structure	SCL structure

See *Structured Text Syntax* for more information on the syntax of expressions within structured text.

SCALE Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If false, the instruction does not execute and outputs are not updated. Default is true.
In	REAL	The analog signal input. Valid = any float Default = 0.0
InRawMax	REAL	The maximum value attainable by the input to the instruction. If InRawMax ≤ InRawMin, the instruction sets the appropriate bit in Status and stops updating the output. Valid = InRawMax > InRawMin Default = 0.0
InRawMin	REAL	The minimum value attainable by the input to the instruction. If InRawMin ≥ InRawMax, the instruction sets the appropriate bit in Status and stops updating the output. Valid = InRawMin < InRawMax Default = 0.0
InEUMax	REAL	The scaled value of the input corresponding to InRawMax. Valid = any real value Default = 0.0
InEUMin	REAL	The scaled value of the input corresponding to InRawMin. Valid = any real value

		Default = 0.0
Limiting	BOOL	Limiting selector. If true, Out is limited to between InEUMin and InEUMax. Default is false.

Output Parameter	Data Type	Description
EnableOut	BOOL	Indicates if instruction is enabled. Cleared to false if Out overflows.
Out	REAL	The output that represents scaled value of the analog input. Valid = any real value Default = InEUMin
MaxAlarm	BOOL	The above maximum input alarm indicator. This value is set to true when In> InRawMax.
MinAlarm	BOOL	The below minimum input alarm indicator. This value is set to true when In < InRawMin.
Status	DINT	Status of the function block.
InstructFault (Status.0)	BOOL	The instruction detected one of the following execution errors. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.
InRawRangeInv (Status.1)	BOOL	InRawMin ≥ InRawMax.

Description

Use the SCL instruction with analog input modules that do not support scaling to a full resolution floating point value.

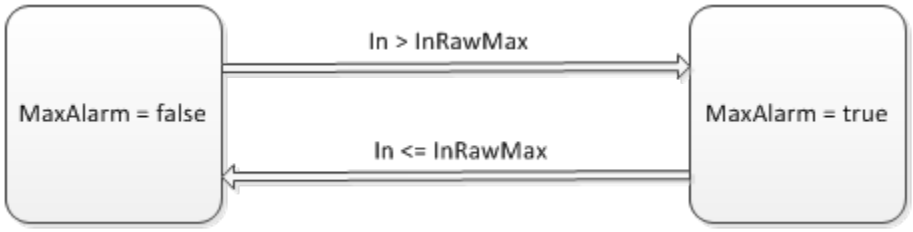
For example, the 1771-IFE module is a 12-bit analog input module that supports scaling only in integer values. If you use a 1771-IFE module to read a flow of 0-100 gallons per minute (gpm), you typically do not scale the module from 0-100 because that limits the resolution of the module. Instead, use the SCL instruction and configure the module to return an unscaled (0-4095) value, which the SCL instruction converts to 0-100 gpm (floating point) without a loss of resolution. This scaled value could then be used as an input to other instructions.

The SCL instruction uses this algorithm to convert unscaled input into a scaled value:

$$Out = (In - InRawMin) \times \left(\frac{InEUMax - InEUMin}{InRawMax - InRawMin} \right) + InEUMin$$

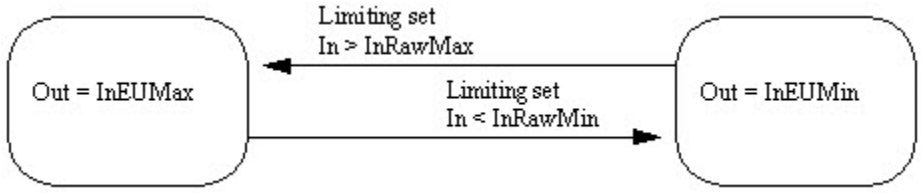
Alarming

Once the instruction calculates Out, the MaxAlarm and MinAlarm are determined as follows:



Limiting

Limiting is performed on Out when Limiting is set. The instruction sets Out = InEUMax when In > InRawMax. The instruction sets Out = InEUMin when In < InRawMin.



Affects Math Status Flags

No

Major/Minor Faults

None specific to this instruction. See Common Attributes on page for operand-related faults.

Execution

Function Block

Condition/State	Action Taken
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is true	EnableIn and EnableOut bits are set to true The instruction executes.
Instruction first run	N/A
Instruction first scan	N/A
Postscan	EnableIn and EnableOut bits are cleared to false.

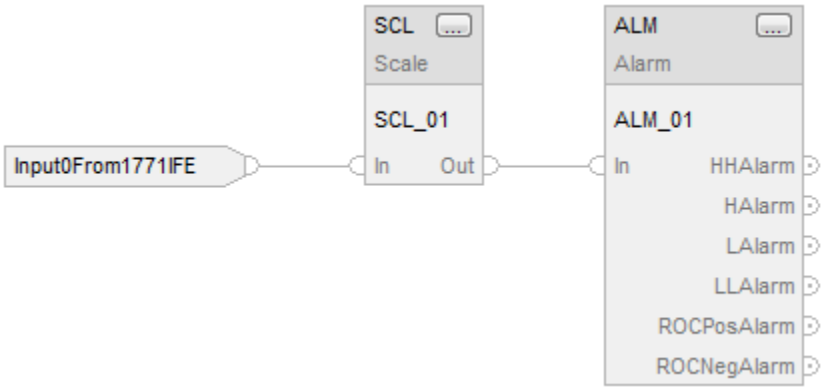
Structured Text

Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.
Normal Execution	See Tag.EnableIn is true in the Function Block table.
Postscan	See Postscan in the Function Block table.

Example

The SCL instruction is typically used with analog input modules that do not support on-board scaling to floating point engineering units. In this example, the SCL instruction scales an analog input from a 1771-IFE module. The instruction places the result in Out, which is used by an ALM instruction.

Function Block



Structured Text

```
SCL_01.In := Input0From1771IFE;
SCL(SCL_01);

ALM_01.In := SCL_01.Out;
ALM(ALM_01);
```

Split Range Time Proportional (SRTP)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, Compact GuardLogix 5380, CompactLogix 5380, ControlLogix 5580, GuardLogix 5580, and ControlLogix 5590 controllers.

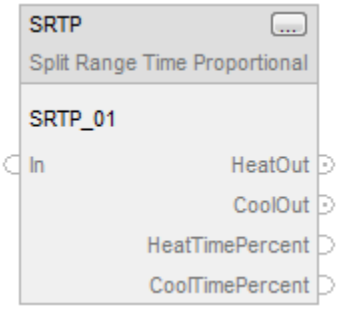
The Split Range Time Proportional (SRTP) instruction takes the 0-100% output of a PID loop and drives heating and cooling digital output contacts with a periodic pulse. This instruction controls applications such as barrel temperature control on extrusion machines.

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram logic.

Function Block



Structured Text

SRTP(SRTP_tag)

Operands

Function Block

Operand	Type	Format	Description
SRTP tag	SPLIT_RANGE	Structure	SRTP structure

Structured Text

Operand	Type	Format	Description
SRTP tag	SPLIT_RANGE	Structure	SRTP structure

See *Structured Text Syntax* for more information on the syntax of expressions within structured text.

SPLIT_RANGE Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If false, the instruction does not execute and outputs are not updated. Default is true.
In	REAL	The analog signal input asking for heating or cooling. This input typically comes from the CVEU of a PID loop.

Input Parameter	Data Type	Description
		Valid = any float
CycleTime	REAL	The period of the output pulses in seconds. A value of zero turns off both heat and cool outputs. If this value is invalid, the instruction assumes a value of zero and sets the appropriate bit in Status. Valid = any positive float Default = 0.0
MaxHeatIn	REAL	Maximum heat input. This value specifies the percentage of the In which will cause maximum heating. This is typically 100% for a heat/cool loop. Valid = any float Default = 100.0
MinHeatIn	REAL	Minimum heat input. Specify the percent of In that represents the start of the heating range and causes minimum heating. This is typically 50% for a heat/cool loop. Valid = any float Default = 50.0
MaxCoolIn	REAL	Maximum cool input. Specify the percent of In that causes maximum cooling. This is typically 0% for a heat/cool loop. Valid = any float Default = 0.0
MinCoolIn	REAL	Minimum cool input. Specify the percent of In that causes minimum cooling. This is typically 50% for a heat/cool loop. Valid = any float Default = 50.0
MaxHeatTime	REAL	Maximum heat time in seconds. Specify the maximum time in seconds that a heating pulse can be on. If the instruction calculates HeatTime to be greater than this value, HeatTime is limited to MaxHeatTime. If MaxHeatTime is invalid, the instruction assumes a value of CycleTime and sets the appropriate bit in Status. Valid = 0.0 to CycleTime Default = CycleTime

Input Parameter	Data Type	Description
MinHeatTime	REAL	Minimum heat time in seconds. Specify the minimum time in seconds that a heating pulse can be on. If the instruction calculates HeatTime to be less than this value, HeatTime is set to zero. If MinHeatTime is invalid, the instruction assumes a value of zero and sets the appropriate bit in Status. Valid = 0.0 to MaxHeatTime Default = 0.0
MaxCoolTime	REAL	Maximum cool time in seconds. Specify the maximum time in seconds that a cooling pulse can be on. If the instruction calculates CoolTime to be larger than this value, CoolTime is limited to MaxCoolTime. If MaxCoolTime is invalid, the instruction assumes a value of CycleTime and sets the appropriate bit in Status. Valid = 0.0 to CycleTime Default = CycleTime
MinCoolTime	REAL	Minimum cool time in seconds. Specify the minimum time in seconds that a cooling pulse can be on. If the instruction calculates CoolTime to be less than this value, CoolTime is set to zero. If MinCoolTime is invalid, the instructions assumes a value of zero and sets the appropriate bit in Status. Valid = 0.0 to MaxCoolTime Default = 0.0

Output Parameter	Data Type	Description
EnableOut	BOOL	Indicates if instruction is enabled. Cleared to false if HeatTimePercent or CoolTimePercent overflows.
HeatOut	BOOL	Heating output pulse. The instruction pulses this output for the heating contact.

Output Parameter	Data Type	Description
CoolOut	BOOL	Cooling output pulse. The instruction pulses this output for the cooling contact.
HeatTimePercent	REAL	Heating output pulse time in percent. This value is the calculated percent of the current cycle that the HeatingOutput will be on. This allows you to use the instruction with an analog output for heating if required.
CoolTimePercent	REAL	Cooling output pulse time in percent. This value is the calculated percent of the current cycle that the CoolingOutput will be on. This allows you to use the instruction with an analog output for cooling if required.
Status	DINT	Status of the function block.
InstructFault (Status.0)	BOOL	The instruction detected one of the following execution errors. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.
CycleTimeInv (Status.1)	BOOL	Invalid CycleTime value. The instruction uses zero.
MaxHeatTimeInv (Status.2)	BOOL	Invalid MaxHeatTime value. The instruction uses the CycleTime value.
MinHeatTimeInv (Status.3)	BOOL	Invalid MinHeatTime value. The instruction uses zero.
MaxCoolTimeInv (Status.4)	BOOL	Invalid MaxCoolTime value. The instruction uses the CycleTime value.
MinCoolTimeInv (Status.5)	BOOL	Invalid MinCoolTime value. The instruction uses zero.
HeatSpanInv (Status.6)	BOOL	MaxHeatIn = MinHeatIn.
CoolSpanInv (Status.7)	BOOL	MaxCoolIn = MinCoolIn.

Description

The length of the SRTP pulse is proportional to the PID output. The instruction parameters accommodate heating and cooling applications.

Using the Internal Cycle Timer

The instruction maintains a free running cycle timer that cycles from zero to the programmed CycleTime. The internal timer is updated by DeltaT. DeltaT is the elapsed time since the instruction last executed. This timer determines if the outputs need to be turned on.

You can change CycleTime at any time. If CycleTime = 0, the internal timer is cleared and HeatOut and CoolOut are cleared to false.

Calculating Heat and Cool Times

Heat and cool times are calculated every time the instruction is executed.

HeatTime is the amount of time within CycleTime that the heat output is to be turned on.

$$HeatTime = \frac{In - MinHeatIn}{MaxHeatIn - MinHeatIn} \times CycleTime$$

If HeatTime < MinHeatTime, set HeatTime = 0.

If HeatTime> MaxHeatTime, limit HeatTime = MaxHeatTime.

HeatTimePercent is the percentage of CycleTime that the HeatOut pulse is true.

$$HeatTimePercent = \frac{HeatTime}{CycleTime} \times 100$$

CoolTime is the amount of time within CycleTime that the cool output is to be turned on.

$$CoolTime = \frac{In - MinCoolIn}{MaxCoolIn - MinCoolIn} \times CycleTime$$

If CoolTime < MinCoolTime, set CoolTime = 0.

If CoolTime > MaxCoolTime, limit CoolTime = MaxCoolTime.

CoolTimePercent is the percentage of CycleTime that the CoolOut pulse is true.

$$CoolTimePercent = \frac{CoolTime}{CycleTime} \times 100$$

The instruction controls heat and cool outputs using these rules:

- Set HeatOut to true if HeatTime ≥ the internal cycle time accumulator. Clear HeatOut to false when the internal cycle timer > HeatTime.
- Set CoolOut to true if CoolTime ≥ the internal cycle time accumulator. Clear CoolOut to false if the internal cycle timer > CoolTime.
- Clear HeatOut and CoolOut to false if CycleTime = 0.

Affects Math Status Flags

No

Major/Minor Faults

None specific to this instruction. See Common Attributes on page for operand-related faults.

Execution

Function Block

Condition/State	Action Taken
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is true	EnableIn and EnableOut bits are set to true. The instruction executes.
Instruction first run	N/A
Instruction first scan	HeatOut and CoolOut are cleared to false. HeatTimePercent and CoolTimePercent are cleared to 0.0.
Postscan	EnableIn and EnableOut bits are cleared to false.

Structured Text

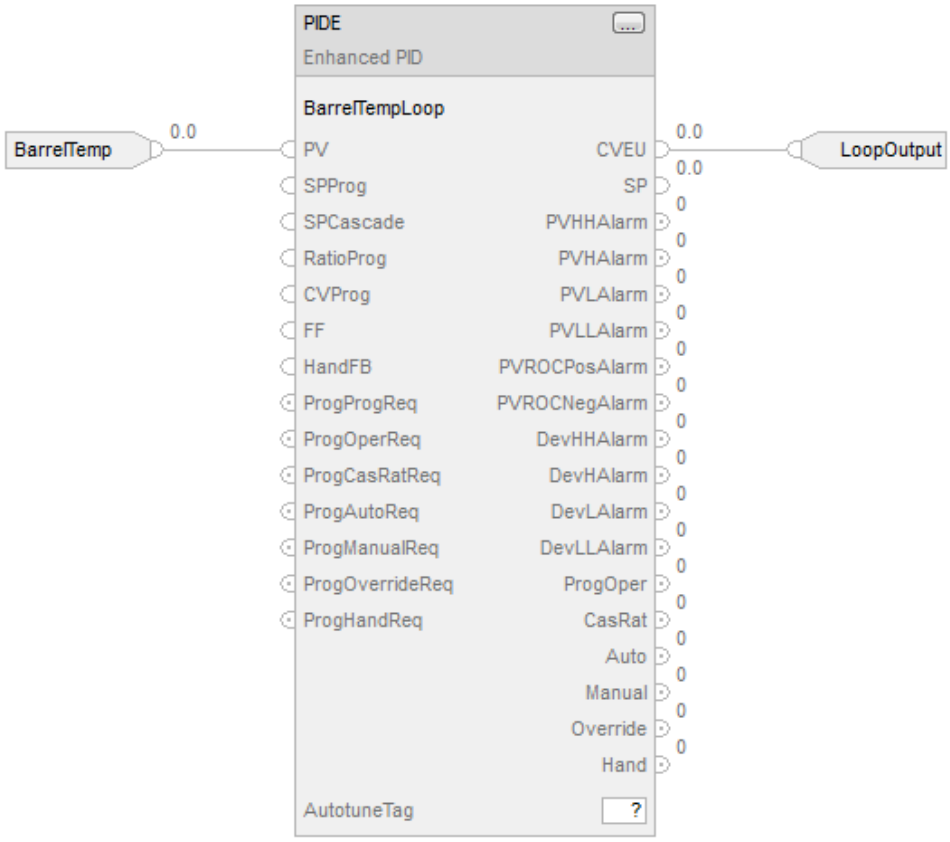
Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.
Normal Execution	See Tag.EnableIn is true in the Function Block table.
Postscan	See Postscan in the Function Block table.

Example

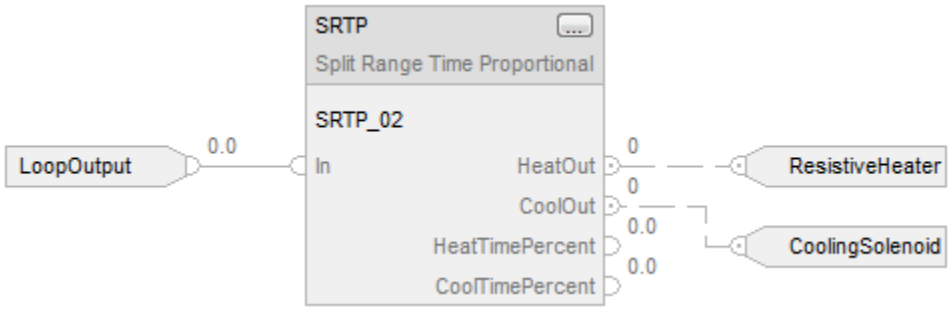
In this example, a PIDE instruction executes in a slow, lower priority task because it is a slow, temperature loop. The output of the PIDE instruction is a controller-scoped tag because it becomes the input to an SRTP instruction. The SRTP instruction executes in a faster, higher priority task so that the pulse outputs are more accurate

Function Block

Place the PIDE instruction in a slow, lower priority task



Place the SRTP instruction in a faster, higher priority task.



Structured Text

Place the PIDE instruction in a slow, lower priority task.

```
BarrelTempLoop.PV := BarrelTemp;
PIDE(BarrelTempLoop);
LoopOutput := BarrelTempLoop.CVEU;
```

Place the SRTP instruction in a faster, higher priority task.

```
SRTP_02.In := LoopOutput;
SRTP(SRTP_02);
```

```
ResistiveHeater := SRTP_02.HeatOut;
CoolingSolenoid := SRTP_02.CoolOut;
```

Totalizer (TOT)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, Compact GuardLogix 5380, CompactLogix 5380, ControlLogix 5580, GuardLogix 5580, and ControlLogix 5590 controllers.

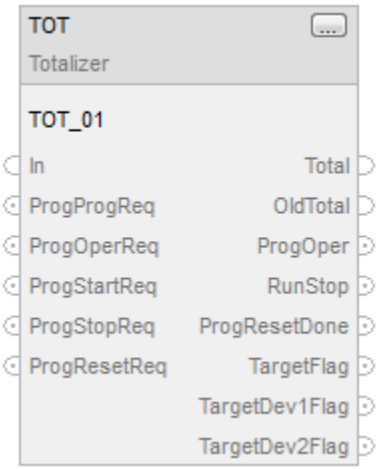
The Totalizer (TOT) instruction provides a time-scaled accumulation of an analog input value.

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram logic.

Function Block



Structured Text

TOT(TOT_tag)

Operands

Function Block

Operand	Type	Format	Description
TOT tag	TOTALIZER	Structure	TOT structure

Structured Text

Operand	Type	Format	Description
TOT tag	TOTALIZER	Structure	TOT structure

See *Structured Text Syntax* for more information on the syntax of expressions within structured text.

TOTALIZER Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If false, the instruction does not execute and outputs are not updated. Default is true.
In	REAL	The analog signal input to the instruction. Valid = any float Default = 0.0
InFault	BOOL	Bad health indicator of In. If true, it indicates the input signal has an error, the instruction sets the appropriate bit in Status, the control algorithm is not executed, and Total is not updated. Default is false.
TimeBase	DINT	The timebase input. The time base of the totalization based on the In engineering units. 0 = Seconds 1 = Minutes 2 = Hours 3 = Days For example, use TimeBase = minutes if In has units of gal/min. If this value is invalid, the instruction sets the appropriate bit in Status and does not update the Total. For more information about timing modes, see Function Block Attributes. Valid = 0 to 3 Default = 0
Gain	REAL	The multiplier of the incremental totalized value. The user can use the Gain to convert the units of totalization. For example, use the Gain to convert gal/min to a total in barrels. Valid = any float Default = 1.0

Input Parameter	Data Type	Description
ResetValue	REAL	The reset value input. The reset value of Total when OperResetReq or ProgResetReq transitions from false to true. Valid = any float Default = 0.0
Target	REAL	The target value for the totalized In. Valid = any float Default = 0.0
TargetDev1	REAL	The large deviation pre-target value of the Total compared to the Target. This value is expressed as a deviation from the Target. Valid = any float Default = 0.0
TargetDev2	REAL	The small deviation pre-target value of the Total compared to the Target. This value is expressed as a deviation from the Target. Valid = any float Default = 0.0
LowInCutoff	REAL	The instruction low input cutoff input. When the In is at or below the LowInCutoff value, totalization ceases. Valid = any float Default = 0.0
ProgProgReq	BOOL	Program program request. Set to true to request Program control. Ignored if ProgOperReq is true. Holding this true and ProgOperReq false locks the instruction in Program control. Default is false.
ProgOperReq	BOOL	Program operator request. Set to true to request Operator control. Holding this true locks the instruction in Operator control. Default is false.
ProgStartReq	BOOL	The program start request input. Set to true to request totalization to start. Default is false.
ProgStopReq	BOOL	The program stop request input. Set to true to request totalization to stop.

Input Parameter	Data Type	Description
		Default is false.
ProgResetReq	BOOL	The program reset request input. Set to true to request the Total to reset to the ResetValue. Default is false.
OperProgReq	BOOL	Operator program request. Set to true by the operator interface to request Program control. The instruction clears this input to false. Default is false.
OperOperReq	BOOL	Operator operator request. Set to true by the operator interface to request Operator control. The instruction clears this input to false. Default is false.
OperStartReq	BOOL	The operator start request input. Set to true by the operator interface to request totalization to start. The instruction clears this input to false. Default is false.
OperStopReq	BOOL	The operator stop request input. Set to true by the operator interface to request totalization to stop. The instruction clears this input to false. Default is false.
OperResetReq	BOOL	The operator reset request input. Set to true by the operator interface to request totalization to reset. The instruction clears this input to false. Default is false.
ProgValueReset	BOOL	Reset program control values. When true, clear all the program request inputs to false at each execution of the instruction. Default is false.
TimingMode	DINT	Selects timing execution mode. 0 = Period mode 1 = Oversample mode 2 = Real time sampling mode For more information about timing modes, see Function Block Attributes. Valid = 0 to 2 Default = 0

Input Parameter	Data Type	Description
OversampleDT	REAL	Execution time for oversample mode. Valid = 0 to 4194.303 seconds Default = 0
RTSTime	DINT	Module update period for real time sampling mode Valid = 1 to 32,767ms Default = 1
RTSTimeStamp	DINT	Module time stamp value for real time sampling mode. Valid = 0 to 32,767ms Default = 0

Output Parameter	Data Type	Description
EnableOut	BOOL	Indicates if instruction is enabled. Cleared to false if Total overflows.
Total	REAL	The totalized value if In.
OldTotal	REAL	The value of the total before a reset occurred. You can monitor this value to read the exact total just before the last reset.
ProgOper	BOOL	Program/operator control indicator. True when in Program control. False when in Operator control.
RunStop	BOOL	The indicator of the operational state of the totalizer. True when the TOT instruction is running. False when the TOT instruction is stopped.
ProgResetDone	BOOL	The indicator that the TOT instruction has completed a program reset request. Set to true when the instruction resets as a result of ProgResetReq. You can monitor this to determine that a reset successfully completed. Cleared to false when ProgResetReq is false.
TargetFlag	BOOL	The flag for Total. Set to true when Total ≥ Target.

Output Parameter	Data Type	Description
TargetDev1Flag	BOOL	The flag for TargetDev1. Set to true when Total ≥ Target - TargetDev1.
TargetDev2Flag	BOOL	The flag for TargetDev2. Set to true when Total ≥ Target - TargetDev2.
LowInCutoffFlag	BOOL	The instruction low input cutoff flag output. Set to true when In £ LowInCutoff.
DeltaT	REAL	Elapsed time between updates. This is the elapsed time in seconds used by the control algorithm to calculate the process output.
Status	DINT	Status of the function block.
InstructFault (Status.0)	BOOL	The instruction detected one of the following execution errors. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.
InFaulted (Status.1)	BOOL	In value faulted.
TimeBaseInv (Status.2)	BOOL	Invalid TimeBase value.
TimingModeInv (Status.27)	BOOL	Invalid TimingMode value.
RTSMissed (Status.28)	BOOL	Only used in real time sampling mode. Set to true when ABS(DeltaT – RTSTime) > 1 millisecond.
RTSTimeInv (Status.29)	BOOL	Invalid RTSTime value.
RTSTimeStampInv (Status.30)	BOOL	Invalid RTSTimeStamp value.
DeltaTInv (Status.31)	BOOL	Invalid DeltaT value. This can occur if OversampleDT is invalid in oversample timing mode.

Description

This instruction typically totals the amount of a material added over time, based on a flow signal.

The TOT instruction supports:

- Time base selectable as seconds, minutes, hours, or days.
- You can specify a target value and up to two pre-target values. Pre-target values are typically used to switch to a slower feed rate. Digital flags announce the reaching of the target or pre-target values.
- A low flow input cutoff that you can use to eliminate negative totalization due to slight flow meter calibration inaccuracies when the flow is shut off.
- Operator or program capability to start/stop/reset.
- A user defined reset value.
- Trapezoidal-rule numerical integration to improve accuracy.
- The internal totalization is done with double precision math to improve accuracy.

Monitoring the TOT Instruction

There is an operator faceplate available for the TOT instruction.

Affects Math Status Flags

No

Major/Minor Faults

None specific to this instruction. See Common Attributes on page for operand-related faults.

Execution

Function Block

Condition/State	Action Taken
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is true	EnableIn and EnableOut bits are set to true. The instruction executes.
Instruction first run	Total is set to ResetValue. OldTotal is cleared to 0.0. ProgOper is cleared to false.
Instruction first scan	All operator request inputs are cleared to false. If ProgValueReset is true then all program request inputs are cleared to false.
Postscan	EnableIn and EnableOut bits are cleared to false.

Structured Text

Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.
Normal Execution	See Tag.EnableIn is true in the Function Block table.
Postscan	See Postscan in the Function Block table.

Check for Low Input Cutoff

If (In <= LowInCutoff), the instruction sets LowInCutoffFlag to true and makes In(n-1) = 0.0.

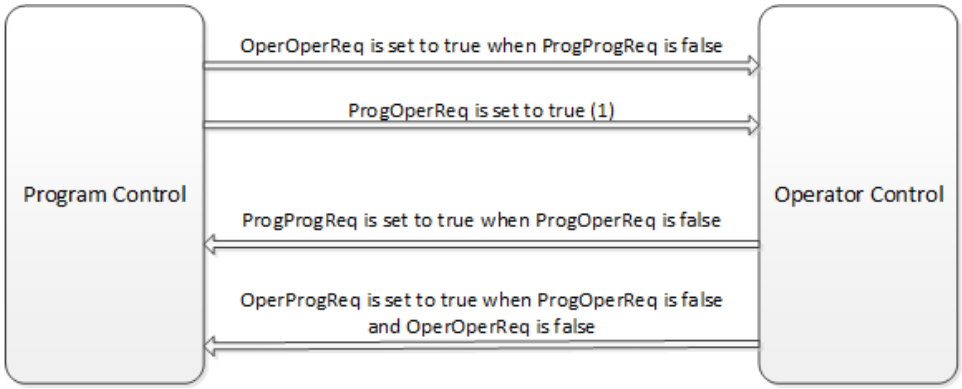
Otherwise, the instruction clears LowInCutoffFlag to false.

When the LowInCutoffFlag is true, the operation mode is determined but totalization ceases.

When LowInCutoffFlag is false, totalization continues that scan.

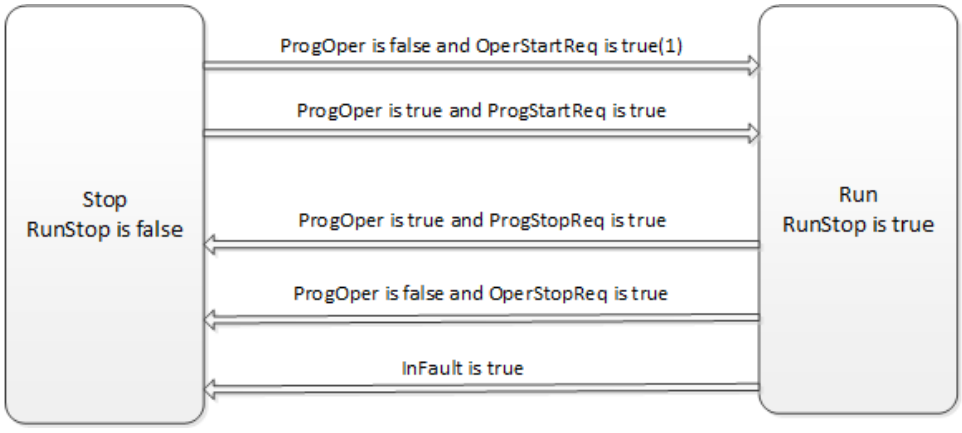
Operating Modes

The following diagram shows how the TOT instruction changes between Program control and Operator control.



(1) The instruction remains in operator control mode when ProgOperReq is true.

The following diagram shows how the TOT instruction changes between Run and Stop modes.



(1) The stop requests take precedence over start requests

(2) The first scan in run after a stop, the totalization is not evaluated, but in $_{n-1}$ is updated.

During the next scan, totalization resumes.

All operator request inputs are cleared to false at the end of each scan. If ProgValueReset is true, all program request inputs are cleared to false at the end of each scan.

Resetting the TOT Instruction

When ProgResetReq transitions to true while ProgOper is true, the following happens:

- OldTotal = Total
- Total = ResetValue
- ProgResetDone is set to true

If ProgResetReq is false and ProgResetDone is true then ProgResetDone is cleared to false

When OperResetReq transitions to true while ProgOper is false, the following happens:

- OldTotal = Total
- Total = ResetValue

Calculating the Totalization

When RunStop is true and LowInCutoffFlag is false, the following equation performs the totalization calculation.

$$Total_n = Total_{n-1} + Gain \times \frac{DeltaT}{2 \times TimeBase} \times (In_n + In_{n-1})$$

where TimeBase is:

Value	Condition
1	TimeBase = 0 (seconds)
60	TimeBase = 1 (minutes)
3600	TimeBase = 2 (hours)
86400	TimeBase = 3 (days)

Determining if Target Values Have Been Reached

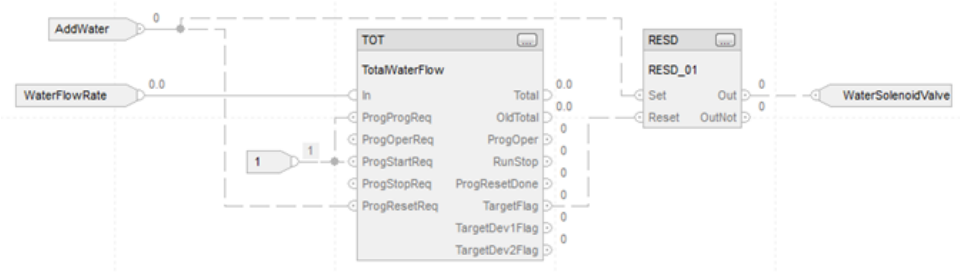
Once the totalization has been calculated, these rules determine whether the target or pre-target values have been reached:

- TargetFlag is true when Total ≥Target
- TargetDev1Flag is true when Total ≥ (Target - TargetDev1)
- TargetDev2Flag is true when Total ≥ (Target - TargetDev2)

Example

In this example, the TOT instruction meters a target quantity of water into a tank and shuts off the flow once the proper amount of water has been added. When the AddWater pushbutton is pressed, the TOT instruction resets and starts totalizing the amount of water flowing into the tank. Once the Target value is reached, the TOT instruction sets the TargetFlag output, which causes the solenoid valve to close. For this example, the TOT instruction was "locked" into Program Run by setting the ProgProgReq and ProgStartReq inputs. This is done for this example because the operator never needs to directly control the TOT instruction.

Function Block



Structured Text

```
TotalWaterFlow.In := WaterFlowRate;
TotalWaterFlow.ProgProgReq := 1;
TotalWaterFlow.ProgStartReq := 1;
TotalWaterFlow.ProgResetReq := AddWater;
TOT(TotalWaterFlow);

RESD_01.Set := AddWater;
RESD_01.Reset := TotalWaterFlow.TargetFlag;
RESD(RESD_01);

WaterSolenoidValve := RESD_01.Out;
```

Coordinated Control (CC)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, Compact GuardLogix 5380, CompactLogix 5380, ControlLogix 5580, GuardLogix 5580, and ControlLogix 5590 controllers.

The Coordinated Control (CC) instruction controls a single process variable by manipulating as many as three different control variables. As an option, any of the three outputs can be used as an input to create feed forward action in the controller. The CC calculates the control variables (CV1, CV2, and CV3) in the auto mode based on the PV - SP deviation, internal models, and tuning.

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram logic.

Function Block

Structured Text

CC(CC_tag);

Operands

Structured Text

Operands	Type	Format	Description
CC tag	COORDINATED_ CONTROL	structure	CC Structure

See Structured Text Syntax for more information on the syntax of expressions within structured text.

IMPORTANT: Whenever an APC block detects a change in Delta Time (DeltaT), a ModelInit will be performed. For this reason the blocks should only be run in one of the TimingModes in which DeltaT will be constant.

- TimingMode = 0 (Periodic) while executing these function blocks in a Periodic Task
- TimingMode = 1 (Oversample)

In either case, if the Periodic Task time is dynamically changed, or the OversampleDT is dynamically changed, the block will perform a ModelInit.

The following TimingMode setting are not recommended due to jitter in DeltaT:

- TimingMode = 0 (Periodic) while executing these function blocks in a Continuous or Event Task
- TimingMode = 2 (RealTimeSample)

Function Block

Operands	Type	Format	Description
CC tag	COORDINATED CONTROL	structure	CC Structure

Structure

Input Parameters	Data Type	Description	Valid and Default Values
EnableIn	BOOL	Enable Input. If False, the function block will not execute and outputs are not updated.	Default=TRUE
PV	REAL	Scaled process variable input. This value is typically read from an analog input module.	Valid = any float Default = 0.0
PVFault	BOOL	PV bad health indicator. If PV is read from an analog input, then PVFault will normally be controlled by the analog input fault status. If PVFault is TRUE, it indicates an error on the input module, set bit in Status.	FALSE = Good Health Default = FALSE
PVEUMax	REAL	Maximum scaled value for PV. The value of PV and SP that corresponds to 100% span of the Process Variable. If PVEUMax ≤ PVEUMin, set bit in Status.	Valid = PVEUMin < PVEUMax ≤ maximum positive float Default = 100.0
PVEUMin	REAL	Minimum scaled value for PV. The value of PV and SP that corresponds to 0% span of the Process Variable.	Valid = maximum negative float ≤ PVEUMin < PVEUMax Default = 0.0

Input Parameters	Data Type	Description	Valid and Default Values
		If PVEUMax ≤ PVEUMin, set bit in Status.	
SPProg	REAL	SP Program value, scaled in PV units. SP is set to this value when the instruction is in Program control.	Valid = SPLLimit to SPHLimit Default = 0.0
SPOper	REAL	SP Operator value, scaled in PV units. SP set to this value when in Operator control. If value of SPProg or SPOper < SPLLimit or > SPHLimit, set bit in Status and limit value used for SP.	Valid = SPLLimit to SPHLimit Default = 0.0
SPHLimit	REAL	SP high limit value, scaled in PV units. If SPHLimit < SPLLimit or SPHLimit > PVEUMax, set bit in Status.	Valid = SPLLimit to PVEUMax Default = 100.0
SPLLimit	REAL	SP low limit value, scaled in PV units. If SPLLimit < PVEUMin, or SPHLimit < SPLLimit, set bit in Status and limit SP by using the value of SPLLimit.	Valid = PVEUMin to SPHLimit Default = 0.0
CV1Fault	BOOL	Control variable 1 bad health indicator. If CV1EU controls an analog output, then CV1Fault will normally come from the analog output's fault status. If CV1Fault is TRUE, it indicates an error on the output module, set bit in Status.	Default = FALSE FALSE = Good Health
CV2Fault	BOOL	Control variable 2 bad health indicator. If CV2EU controls an analog output, then CV2Fault will normally come from the analog output's fault status. If CV2Fault is TRUE, it indicates an error on the	Default = FALSE FALSE = Good Health

Input Parameters	Data Type	Description	Valid and Default Values
		output module, set bit in Status.	
CV3Fault	BOOL	Control variable 3 bad health indicator. If CV3EU controls an analog output, then CV3Fault will normally come from the analog output's fault status. If CV3Fault is TRUE, it indicates an error on the output module, set bit in Status.	Default = FALSE FALSE = Good Health
CV1InitReq	BOOL	CV1 initialization request. While TRUE, set CV1EU to the value of CV1InitValue. This signal will normally be controlled by the In Hold status on the analog output module controlled by CV1EU or from the InitPrimary output of a secondary loop.	Default = FALSE
CV2InitReq	BOOL	CV2 initialization request. While TRUE, set CV2EU to the value of CV2InitValue. This signal will normally be controlled by the In Hold status on the analog output module controlled by CV2EU or from the InitPrimary output of a secondary loop.	Default = FALSE
CV3InitReq	BOOL	CV3 initialization request. While TRUE, set CV3EU to the value of CV3InitValue. This signal will normally be controlled by the In Hold status on the analog output module controlled by CV32EU or from the InitPrimary output of a secondary loop.	Default = FALSE
CV1InitValue	REAL	CV1EU initialization value, scaled in CV1EU units. When CV1Initializing is TRUE set CV1EU equal to	Valid = any float Default = 0.0

Input Parameters	Data Type	Description	Valid and Default Values
		CV1InitValue and CV1 to the corresponding percentage value. CV1InitValue will normally come from the feedback of the analog output controlled by CV1EU or from the setpoint of a secondary loop. The instruction initialization is disabled when CVFaulted or CVEUSpanInv are TRUE.	
CV2InitValue	REAL	CV2EU initialization value, scaled in CV2EU units. When CV2Initializing is TRUE set CV2EU equal to CV2InitValue and CV2 to the corresponding percentage value. CV2InitValue will normally come from the feedback of the analog output controlled by CV2EU or from the setpoint of a secondary loop. The instruction initialization is disabled when CVFaulted or CVEUSpanInv are TRUE.	Valid = any float Default = 0.0
CV3InitValue	REAL	CV3EU initialization value, scaled in CV3EU units. When CV3Initializing is TRUE set CV3EU equal to CV3InitValue and CV3 to the corresponding percentage value. CV3InitValue will normally come from the feedback of the analog output controlled by CV3EU or from the setpoint of a secondary loop. The instruction initialization is disabled when CVFaulted or CVEUSpanInv are TRUE.	Valid = any float Default = 0.0

Input Parameters	Data Type	Description	Valid and Default Values
CV1Prog	REAL	CV1 Program-Manual value. CV1 is set to this value when in Program control and Manual mode.	Valid = 0.0 through 100.0 Default = 0.0
CV2Prog	REAL	CV2 Program-Manual value. CV2 is set to this value when in Program control and Manual mode.	Valid = 0.0...100.0 Default = 0.0
CV3Prog	REAL	CV3 Program-Manual value. CV3 is set to this value when in Program control and Manual mode.	Valid = 0.0...100.0 Default = 0.0
CV1Oper	REAL	CV1 Operator-Manual value. CV1 is set to this value when in Operator control and Manual mode. If not Operator-Manual mode, set CV1Oper to the value of CV1 at the end of each function block execution. If value of CV1Oper < 0 or > 100, or < CV1LLimit or > CV1HLimit when CVManLimiting is TRUE, set unique Status bit and limit value used for CV.	Valid = 0.0...100.0 Default = 0.0
CV2Oper	REAL	CV2 Operator-Manual value. CV2 is set to this value when in Operator control and Manual mode. If not Operator-Manual mode, set CV2Oper to the value of CV2 at the end of each function block execution. If value of CV2Oper < 0 or > 100, or < CV2LLimit or > CV2HLimit when CVManLimiting is TRUE, set unique Status bit and limit value used for CV.	Valid = 0.0...100.0 Default = 0.0
CV3Oper	REAL	CV3 Operator-Manual value. CV3 is set to this value when in Operator control and Manual mode. If not Operator-Manual	Valid = 0.0...100.0 Default = 0.0

Input Parameters	Data Type	Description	Valid and Default Values
		mode, set CV3Oper to the value of CV3 at the end of each function block execution. If value of CV3Oper < 0 or > 100, or < CV3LLimit or > CV3HLimit when CVManLimiting is TRUE, set unique Status bit and limit value used for CV.	
CV1OverrideValue	REAL	CV1 Override value. CV1 set to this value when in Override mode. This value should correspond to a safe state output of the loop. If value of CV1OverrideValue < 0 or >100, set unique Status bit and limit value used for CV.	Valid = 0.0...100.0 Default = 0.0
CV2OverrideValue	REAL	CV2 Override value. CV2 set to this value when in Override mode. This value should correspond to a safe state output of the loop. If value of CV2OverrideValue < 0 or >100, set unique Status bit and limit value used for CV.	Valid = 0.0...100.0 Default = 0.0
CV3OverrideValue	REAL	CV3 Override value. CV3 set to this value when in Override mode. This value should correspond to a safe state output of the loop. If value of CV3OverrideValue < 0 or >100, set unique Status bit and limit value used for CV.	Valid = 0.0...100.0 Default = 0.0
CV1TrackValue	REAL	CV1 track value. When CVTrackReq is enabled and the CC function block is in Manual mode, the CV1TrackValue will	Valid = 0.0...100.0 Default = 0.0

Input Parameters	Data Type	Description	Valid and Default Values
		be ignored, and the CC internal model will update its historical data with the CV1Oper or CV1Prog value. When CVTrackReq is enabled and the CC function block is in Auto, the internal model will update its historical data based on the value of CV1TrackValue. The CV1 in this case will be allowed to move as if the CC function block was still controlling the process. This is useful in multiloop selection schemes where you want the CC function block to follow the output of a different controlling algorithm, where you would connect the output of the controlling algorithm into the CV1TrackValue.	
CV2TrackValue	REAL	CV2 track value. When CVTrackReq is enabled and the CC function block is in Manual mode, the CV2TrackValue will be ignored, and the CC internal model will update its historical data with the CV2Oper or CV2Prog value. When CVTrackReq is enabled and the CC function block is in Auto, the internal model will update its historical data based on the value of CV2TrackValue. The CV2 in this case will be allowed to move as if the CC function block was still controlling the process. This is useful in multiloop selection schemes where you want the CC function block to follow the output of a different controlling	Valid = 0.0...100.0 Default = 0.0

Input Parameters	Data Type	Description	Valid and Default Values
		algorithm, where you would connect the output of the controlling algorithm into the CV2TrackValue.	
CV3TrackValue	REAL	CV3 track value. When CVTrackReq is enabled and the CC function block is in Manual mode, the CV3TrackValue will be ignored, and the CC internal model will update its historical data with the CV3Oper or CV3Prog value. When CVTrackReq is enabled and the CC function block is in Auto, the internal model will update its historical data based on the value of CV3TrackValue. The CV3 in this case will be allowed to move as if the CC function block was still controlling the process. This is useful in multiloop selection schemes where you want the CC function block to follow the output of a different controlling algorithm, where you would connect the output of the controlling algorithm into the CV3TrackValue.	Valid = 0.0...100.0 Default = 0.0
CVManLimiting	BOOL	Limit CV(n), where (n) can be 1, 2, or 3, in Manual mode request. If Manual mode and CVManLimiting is TRUE, CV will be limited by the CV(n)HLimit and CV(n)LLimit values.	Default = FALSE
CV1EUMax	REAL	Maximum value for CV1EU. The value of CV1EU that corresponds to 100% CV1. If CV1EUMax = CV1EUMin, set bit in Status.	Valid = any float Default = 100.0
CV2EUMax	REAL	Maximum value for CV2EU. The value of CV2EU that	Valid = any float

Input Parameters	Data Type	Description	Valid and Default Values
		corresponds to 100% CV2. If CV2EUMax = CV2EUMin, set bit in Status.	Default = 100.0
CV3EUMax	REAL	Maximum value for CV3EU. The value of CV3EU that corresponds to 100% CV3. If CV3EUMax = CV3EUMin, set bit in Status.	Valid = any float Default = 100.0
CV1EUMin	REAL	Minimum value of CV1EU. The value of CV1EU that corresponds to 0% CV1. If CV1EUMax = CV1EUMin, set bit in Status.	Valid = any float Default = 0.0
CV2EUMin	REAL	Minimum value of CV2EU. The value of CV2EU that corresponds to 0% CV2. If CV2EUMax = CV2EUMin, set bit in Status.	Valid = any float Default = 0.0
CV3EUMin	REAL	Minimum value of CV3EU. The value of CV3EU that corresponds to 0% CV3. If CV3EUMax = CV3EUMin, set bit in Status.	Valid = any float Default = 0.0
CV1HLimit	REAL	CV1 high limit value. This is used to set the CV1HAlarm output. It is also used for limiting CV1 when in Auto mode or in Manual if CVManLimiting is TRUE. If CV1HLimit > 100, if CV1HLimit < CV1LLimit, set bit in Status. If CV1HLimit < CV1LLimit, limit CV1 by using the value of CV1LLimit.	Valid = CV1LLimit < CV1HLimit ≤ 100.0 Default = 100.0
CV2HLimit	REAL	CV2 high limit value. This is used to set the CV2HAlarm output. It is also used for limiting CV2 when in Auto mode or in Manual if CVManLimiting is TRUE. If CV2HLimit > 100, if CV2HLimit < CV2LLimit, set bit in Status.	Valid = CV2LLimit < CV2HLimit ≤ 100.0 Default = 100.0

Input Parameters	Data Type	Description	Valid and Default Values
		If CV2HLimit < CV2LLimit, limit CV2 by using the value of CV2LLimit.	
CV3HLimit	REAL	CV3 high limit value. This is used to set the CV3HAlarm output. It is also used for limiting CV3 when in Auto mode or in Manual if CVManLimiting is TRUE. If CV3HLimit > 100, if CV3HLimit < CV3LLimit, set bit in Status. If CV3HLimit < CV3LLimit, limit CV3 by using the value of CV3LLimit.	Valid = CV3LLimit < CV3HLimit ≤ 100.0 Default = 100.0
CV1LLimit	REAL	CV1 low limit value. This is used to set the CV1LAlarm output. It is also used for limiting CV1 when in Auto mode or in Manual mode if CVManLimiting is TRUE. If CV1LLimit < 0 set bit in Status. If CV1HLimit < CV1LLimit, limit CV by using the value of CV1LLimit.	Valid = 0.0 ≤ CV1LLimit < CV1HLimit Default = 0.0
CV2LLimit	REAL	CV2 low limit value. This is used to set the CV2LAlarm output. It is also used for limiting CV2 when in Auto mode or in Manual mode if CVManLimiting is TRUE. If CV2LLimit < 0 set bit in Status. If CV2HLimit < CV2LLimit, limit CV by using the value of CV2LLimit.	Valid = 0.0 ≤ CV2LLimit < CV1HLimit Default = 0.0
CV3LLimit	REAL	CV3 low limit value. This is used to set the CV3LAlarm output. It is also used for limiting CV3 when in Auto mode or in Manual mode if CVManLimiting is TRUE. If CV3LLimit < 0 set bit in Status. If CV3HLimit < CV3LLimit, limit CV	Valid = 0.0 ≤ CV3LLimit < CV1HLimit Default = 0.0

Input Parameters	Data Type	Description	Valid and Default Values
		by using the value of CVLLimit.	
CV1ROCPosLimit	REAL	CV1 rate of change limit, in percent per second. Rate of change limiting is only used when in Auto mode or in Manual mode if CVManLimiting is TRUE. A value of zero disables CV1 ROC limiting. If value of CV1ROCLimit < 0, set bit in Status and disable CV1 ROC limiting.	Valid = 0.0 to maximum positive float Default = 0.0
CV2ROCPosLimit	REAL	CV2 rate of change limit, in percent per second. Rate of change limiting is only used when in Auto mode or in Manual mode if CVManLimiting is TRUE. A value of zero disables CV2 ROC limiting. If value of CV2ROCLimit < 0, set bit in Status and disable CV2 ROC limiting.	Valid = 0.0 to maximum positive float Default = 0.0
CV3ROCPosLimit	REAL	CV3 rate of change limit, in percent per second. Rate of change limiting is only used when in Auto mode or in Manual mode if CVManLimiting is TRUE. A value of zero disables CV3 ROC limiting. If value of CV3ROCLimit < 0, set bit in Status and disable CV3 ROC limiting.	Valid = 0.0 to maximum positive float Default = 0.0
CV1ROCNegLimit	REAL	CV1 rate of change limit, in percent per second. Rate of change limiting is only used when in Auto mode or in Manual mode if CVManLimiting is TRUE. A value of zero disables CV1 ROC limiting. If value of CV1ROCLimit < 0, set bit in Status and disable CV1 ROC limiting.	Valid = 0.0 to maximum positive float Default = 0.0

Input Parameters	Data Type	Description	Valid and Default Values
CV2ROCNegLimit	REAL	CV2 rate of change limit, in percent per second. Rate of change limiting is only used when in Auto mode or in Manual mode if CVManLimiting is TRUE. A value of zero disables CV2 ROC limiting. If value of CV2ROCLimit < 0, set bit in Status and disable CV2 ROC limiting.	Valid = 0.0 to maximum positive float Default = 0.0
CV3ROCNegLimit	REAL	CV3 rate of change limit, in percent per second. Rate of change limiting is only used when in Auto mode or in Manual mode if CVManLimiting is TRUE. A value of zero disables CV3 ROC limiting. If value of CV3ROCLimit < 0, set bit in Status and disable CV3 ROC limiting.	Valid = 0.0 to maximum positive float Default = 0.0
CV1HandFB	REAL	CV1 HandFeedback value. CV1 set to this value when in Hand mode and CV1HandFBFault is FALSE (good health). This value would typically come from the output of a field mounted hand/auto station and would be used to generate a bumpless transfer out of Hand mode. If value of CV1HandFB < 0 or > 100, set unique Status bit and limit value used for CV1.	Valid = 0.0...100.0 Default = 0.0
CV2HandFB	REAL	CV2 HandFeedback value. CV2 set to this value when in Hand mode and CV2HandFBFault is FALSE (good health). This value would typically come from the output of a field mounted hand/auto station and would be used to generate a bumpless transfer out of Hand mode.	Valid = 0.0...100.0 Default = 0.0

Input Parameters	Data Type	Description	Valid and Default Values
		If value of CV2HandFB < 0 or > 100, set unique Status bit and limit value used for CV2.	
CV3HandFB	REAL	CV3 HandFeedback value. CV3 set to this value when in Hand mode and CV3HandFBFault is FALSE (good health). This value would typically come from the output of a field mounted hand/auto station and would be used to generate a bumpless transfer out of Hand mode. If value of CV3HandFB < 0 or > 100, set unique Status bit and limit value used for CV3.	Valid = 0.0...100.0 Default = 0.0
CV1HandFBFault	BOOL	CV1HandFB value bad health indicator. If the CV1HandFB value is read from an analog input, then CV1HandFBFault will normally be controlled by the status of the analog input channel. If CV1HandFBFault is TRUE, it indicates an error on the input module, set bit in Status.	FALSE = Good Health Default = FALSE
CV2HandFBFault	BOOL	CV2HandFB value bad health indicator. If the CV2HandFB value is read from an analog input, then CV2HandFBFault will normally be controlled by the status of the analog input channel. If CV2HandFBFault is TRUE, it indicates an error on the input module, set bit in Status.	FALSE = Good Health Default = FALSE
CV3HANDFBFault	BOOL	CV3HandFB value bad health indicator. If the CV3HandFB value is read from an analog input,	FALSE = Good Health Default = FALSE

Input Parameters	Data Type	Description	Valid and Default Values
		then CV3HandFBFault will normally be controlled by the status of the analog input channel. If CV3HandFBFault is TRUE, it indicates an error on the input module, set bit in Status.	
CV1Target	REAL	Target value for CV1.	Valid = 0.0...100.0 Default = 0.0
CV2Target	REAL	Target value for CV2.	Valid = 0.0...100.0 Default = 0.0
CV3Target	REAL	Target value for CV3.	Valid = 0.0...100.0 Default = 0.0
CV1WindupHIn	BOOL	CV1Windup high request. When TRUE, CV1 will not be allowed to increase in value. This signal will typically be the CV1WindupHOut output from a secondary loop.	Default = FALSE
CV2WindupHIn	BOOL	CV2Windup high request. When TRUE, CV2 will not be allowed to increase in value. This signal will typically be the CV2WindupHOut output from a secondary loop.	Default = FALSE
CV3WindupHIn	BOOL	CV3Windup high request. When TRUE, CV3 will not be allowed to increase in value. This signal will typically be the CV3WindupHOut output from a secondary loop.	Default = FALSE
CV1WindupLIn	BOOL	CV1 Windup low request. When TRUE, CV1 will not be allowed to decrease in value. This signal will typically be the CV1WindupLOut output from a secondary loop.	Default = FALSE

Input Parameters	Data Type	Description	Valid and Default Values
CV2WindupLIn	BOOL	CV2 Windup low request. When TRUE, CV2 will not be allowed to decrease in value. This signal will typically be the CV2WindupLOut output from a secondary loop.	Default = FALSE
CV3WindupLIn	BOOL	CV3 Windup low request. When TRUE, CV3 will not be allowed to decrease in value. This signal will typically be the CV3WindupLOut output from a secondary loop.	Default = FALSE
GainEUSpan	BOOL	CVxModelGain units can be represented as EU or % of span. Set to interpret ModelGain as EU, reset to interpret ModelGain as % of Span.	Default = 0
CV1ProcessGainSign	BOOL	Used only for Autotuning. Sign of the process gain (Delta PV/Delta CV1). Set to indicate a negative process gain (increase in output causes a decrease in PV). Reset to indicate a positive process gain (increase in output causes an increase in PV).	Default = FALSE
CV2ProcessGainSign	BOOL	Used only for Autotuning. Sign of the process gain (Delta PV/Delta CV2). Set to indicate a negative process gain (increase in output causes a decrease in PV). Reset to indicate a positive process gain (increase in output causes an increase in PV).	Default = FALSE
CV3ProcessGainSign	BOOL	Used only for Autotuning. Sign of the process gain (Delta PV/Delta CV3).	Default = FALSE

Input Parameters	Data Type	Description	Valid and Default Values
		Set to indicate a negative process gain (increase in output causes a decrease in PV). Reset to indicate a positive process gain (increase in output causes an increase in PV).	
ProcessType	DINT	Process type selection (1=Integrating, 0=non-integrating)	Default = 0
CV1ModelGain	REAL	The internal model gain parameter for CV1. Enter a positive or negative gain depending on process direction.	Valid = maximum negative float –> maximum positive float Default = 0.0
CV2ModelGain	REAL	The internal model gain parameter for CV2. Enter a positive or negative gain depending on process direction.	Valid = maximum negative float –> maximum positive float Default = 0.0
CV3ModelGain	REAL	The internal model gain parameter for CV3. Enter a positive or negative gain depending on process direction.	Valid = maximum negative float –> maximum positive float Default = 0.0
CV1ModelTC	REAL	The internal model time constant for CV1 in seconds.	Valid = 0.0 to maximum positive float Default = 0.0
CV2ModelTC	REAL	The internal model time constant for CV2 in seconds.	Valid = 0.0 to maximum positive float Default = 0.0
CV3ModelTC	REAL	The internal model time constant for CV3 in seconds.	Valid = 0.0 to maximum positive float Default = 0.0
CV1ModelDT	REAL	The internal model deadtime for CV1 in seconds.	Valid = 0.0 to maximum positive float Default = 0.0
CV2ModelDT	REAL	The internal model deadtime for CV2 in seconds.	Valid = 0.0 to maximum positive float

Input Parameters	Data Type	Description	Valid and Default Values
			Default = 0.0
CV3ModelDT	REAL	The internal model deadtime for CV3 in seconds.	Valid = 0.0 to maximum positive float Default = 0.0
CV1RespTC	REAL	The tuning parameter that determines the speed of the control variable action for CV1 in seconds.	Valid = 0.0 to maximum positive float Default = 0.0
CV2RespTC	REAL	The tuning parameter that determines the speed of the control variable action for CV2 in seconds.	Valid = 0.0 to maximum positive float Default = 0.0
CV3RespTC	REAL	The tuning parameter that determines the speed of the control variable action for CV3 in seconds.	Valid = 0.0 to maximum positive float Default = 0.0
Act1stCV	DINT	The first CV to act to compensate for PV-SP deviation. 1=CV1, 2=CV2, 3=CV3	Valid = 1-3 Default = 1
Act2ndCV	DINT	The second CV to act to compensate for PV-SP deviation. 1=CV1, 2=CV2, 3=CV3	Valid = 1-3 Default = 2
Act3rdCV	DINT	The third CV to act to compensate for PV-SP deviation. 1=CV1, 2=CV2, 3=CV3	Valid = 1-3 Default = 3
Target1stCV	DINT	The CV with top priority to be driven to its target value. 1=CV1, 2=CV2, 3=CV3	Valid = 1-3 Default = 1
Target2ndCV	DINT	The CV with second highest priority to be driven to its target value. 1=CV1, 2=CV2, 3=CV3	Valid = 1-3 Default = 2
Target3rdCV	DINT	The CV with third highest priority to be driven to its target value.	Valid = 1-3 Default = 3

Input Parameters	Data Type	Description	Valid and Default Values
		1=CV1, 2=CV2, 3=CV3	
PVTracking	BOOL	SP track PV request. Set TRUE to enable SP to track PV. Ignored when in Auto modes. SP will only track PV when all three outputs are in manual. As soon as any output returns to Auto, PVTracking stops.	Default = FALSE
CVTrackReq	BOOL	CV Track request. Set true to enable CV Tracking when autotune is OFF. Ignored in Hand and Override mode.	Default = FALSE
ManualAfterInit	BOOL	Manual mode after initialization request. When TRUE, the appropriate CV(n), where (n) can be 1, 2, or 3, will be placed in Manual mode when CV(n)Initializing is set TRUE unless the current mode is Override or Hand. When ManualAfterInit is FALSE, the CV(n) mode will not be changed.	Default = FALSE
ProgProgReq	BOOL	Program Program Request. Set TRUE by the user program to request Program control. Ignored if ProgOperReq is TRUE. Holding this TRUE and ProgOperReq FALSE can be used to lock the function block into program control. When ProgValueReset is TRUE, the function block resets the input to FALSE.	Default = FALSE
ProgOperReq	BOOL	Program Operator Request. Set TRUE by the user program to request Operator control.	Default = FALSE

Input Parameters	Data Type	Description	Valid and Default Values
		Holding this TRUE can be used to lock the function block into operator control. If value of HandFB < 0 or > 100, set unique Status bit and limit value used for CV.	
ProgCV1AutoReq	BOOL	Program-Auto mode request for CV1. Set TRUE by the user program to request Auto mode. If value of CV1HandFB < 0 or > 100, set unique Status bit and limit value used for CV1.	Default = FALSE
ProgCV2AutoReq	BOOL	Program-Auto mode request for CV2. Set TRUE by the user program to request Auto mode. If value of CV2HandFB < 0 or > 100, set unique Status bit and limit value used for CV2.	Default = FALSE
ProgCV3AutoReq	BOOL	Program-Auto mode request for CV3. Set TRUE by the user program to request Auto mode. If value of CV3HandFB < 0 or > 100, set unique Status bit and limit value used for CV3.	Default = FALSE
ProgCV1ManualReq	BOOL	Program-Manual mode request for CV1. Set TRUE by the user program to request Manual mode. If value of CV1HandFB < 0 or > 100, set unique Status bit and limit value used for CV1.	Default = FALSE

Input Parameters	Data Type	Description	Valid and Default Values
ProgCV2ManualReq	BOOL	Program-Manual mode request for CV2. Set TRUE by the user program to request Manual mode. If value of CV2HandFB < 0 or > 100, set unique Status bit and limit value used for CV2.	Default = FALSE
ProgCV3ManualReq	BOOL	Program-Manual mode request for CV3. Set TRUE by the user program to request Manual mode. If value of CV3HandFB < 0 or > 100, set unique Status bit and limit value used for CV3.	Default = FALSE
ProgCV1OverrideReq	BOOL	Program-Override mode request for CV1. Set TRUE by the user program to request Override mode. If value of CV1HandFB < 0 or > 100, set unique Status bit and limit value used for CV1.	Default = FALSE
ProgCV2OverrideReq	BOOL	Program-Override mode request for CV2. Set TRUE by the user program to request Override mode. If value of CV2HandFB < 0 or > 100, set unique Status bit and limit value used for CV2.	Default = FALSE
ProgCV3OverrideReq	BOOL	Program-Override mode request for CV3. Set TRUE by the user program to request Override mode. If value of CV3HandFB < 0 or > 100, set unique Status	Default = FALSE

Input Parameters	Data Type	Description	Valid and Default Values
		bit and limit value used for CV3.	
ProgCV1HandReq	BOOL	Program-Hand mode request for CV1. Set TRUE by the user program to request Hand mode. This value will usually be read as a digital input from a hand/auto station.	Default = FALSE
ProgCV2HandReq	BOOL	Program-Hand mode request for CV2. Set TRUE by the user program to request Hand mode. This value will usually be read as a digital input from a hand/auto station.	Default = FALSE
ProgCV3HandReq	BOOL	Program-Hand mode request for CV3. Set TRUE by the user program to request Hand mode. This value will usually be read as a digital input from a hand/auto station.	Default = FALSE
OperProgReq	BOOL	Operator Program Request. Set TRUE by the operator interface to request Program control. The function block resets this parameter to FALSE.	Default = FALSE
OperOperReq	BOOL	Operator Operator Request. Set TRUE by the operator interface to request Operator control. The function block will reset this parameter to FALSE.	Default = FALSE
OperCV1AutoReq	BOOL	Operator-Auto mode request for CV1. Set TRUE by the operator interface to request Auto mode. The function block	Default = FALSE

Input Parameters	Data Type	Description	Valid and Default Values
		will reset this parameter to FALSE.	
OperCV2AutoReq	BOOL	Operator-Auto mode request for CV2. Set TRUE by the operator interface to request Auto mode. The function block will reset this parameter to FALSE.	Default = FALSE
OperCV3AutoReq	BOOL	Operator-Auto mode request for CV3. Set TRUE by the operator interface to request Auto mode. The function block will reset this parameter to FALSE.	Default = FALSE
OperCV1ManualReq	BOOL	Operator-Manual mode request for CV1. Set TRUE by the operator interface to request Manual mode. The function block resets this parameter to FALSE.	Default = FALSE
OperCV2ManualReq	BOOL	Operator-Manual mode request for CV2. Set TRUE by the operator interface to request Manual mode. The function block resets this parameter to FALSE.	Default = FALSE
OperCV3ManualReq	BOOL	Operator-Manual mode request for CV3. Set TRUE by the operator interface to request Manual mode. The function block resets this parameter to FALSE.	Default = FALSE
ProgValueReset	BOOL	Reset Program control values. When TRUE, the Prog_xxx_Req inputs are reset to FALSE.	Default = FALSE

Input Parameters	Data Type	Description	Valid and Default Values
		When TRUE and Program control, set SPProg equal to SP and CVxProg equal to CVx. When ProgValueReset is TRUE, the instruction sets this input to FALSE.	
TimingMode	DINT	Selects Time Base Execution mode. Value/Description 0 = Periodic mode 1 = Oversample mode 2 = Real time sampling mode For more information about timing modes, see Function Block Attributes.	Valid = 0...2 Default = 0
OverSampleDT	REAL	Execution time for Oversample mode.	Valid = 0...4194.303 seconds Default = 0
RTSTime	DINT	Module update period for Real Time Sampling mode.	Valid = 1...32,767 1 count = 1 ms
RTSTimeStamp	DINT	Module time stamp value for Real Time Sampling mode.	Valid = 0...32,767 (wraps from 32,767...0) 1 count = 1 ms
PVTuneLimit	REAL	PV tuning limit scaled in the PV units. When Autotune is running and predicted PV exceeds this limit, the tuning will be aborted.	Valid = any float Default=0
AtuneTimeLimit	REAL	Maximum time for autotune to complete following the CV step change. When autotune exceeds this time, tuning will be aborted.	Valid range: any float > 0. Default = 60 minutes
NoiseLevel	DINT	An estimate of the noise level expected on the PV to compensate for it during tuning.	Range: 0...2 Default=1

Input Parameters	Data Type	Description	Valid and Default Values
		The selections are: 0=low, 1=medium, 2=high	
CV1StepSize	REAL	CV1 step size in percent for the tuning step test. Step size is directly added to CV1 subject to high/low limiting.	Range: -100% ... 100% Default=10%
CV2StepSize	REAL	CV2 step size in percent for the tuning step test. Step size is directly added to CV2 subject to high/low limiting.	Range: -100% ... 100% Default=10%
CV3StepSize	REAL	CV3 step size in percent for the tuning step test. Step size is directly added to CV3 subject to high/low limiting.	Range: -100% ... 100% Default=10%
CV1ResponseSpeed	DINT	Desired speed of closed loop response for CV1. Slow response: ResponseSpeed=0 Medium response: ResponseSpeed=1 Fast response: ResponseSpeed=2. If ResponseSpeed is less than 0, Slow response is used. If ResponseSpeed is greater than 2, Fast response is used.	Range: 0...2 Default=1
CV2ResponseSpeed	DINT	Desired speed of closed loop response for CV2. Slow response: ResponseSpeed=0 Medium response: ResponseSpeed=1 Fast response: ResponseSpeed=2. If ResponseSpeed is less than 0, Slow response is used. If ResponseSpeed is greater than 2, Fast response is used.	Range: 0...2 Default=1

Input Parameters	Data Type	Description	Valid and Default Values
CV3ResponseSpeed	DINT	Desired speed of closed loop response for CV3. Slow response: ResponseSpeed=0 Medium response: ResponseSpeed=1 Fast response: ResponseSpeed=2. If ResponseSpeed is less than 0, Slow response is used. If ResponseSpeed is greater than 2, Fast response is used.	Range: 0...2 Default=1
CV1ModelInit	BOOL	Internal model initialization switch for CV1. Refer to topic CC Function Block Model Initialization in CC Function Block Tuning.	Default = FALSE
CV2ModelInit	BOOL	Internal model initialization switch for CV2. Refer to topic CC Function Block Model Initialization in CC Function Block Tuning.	Default = FALSE
CV3ModelInit	BOOL	Internal model initialization switch for CV3. Refer to topic CC Function Block Model Initialization in CC Function Block Tuning.	Default = FALSE
Factor	REAL	Non-integrating model approximation factor. Only used for integrating process types.	Default = 100
AtuneCV1Start	BOOL	Start Autotune request for CV1. Set True to initiate auto tuning of the CV1 output. Ignored when CV1 is not in Manual mode. The function block resets the input to FALSE.	Default = FALSE
AtuneCV2Start	BOOL	Start Autotune request for CV2. Set True to initiate	Default = FALSE

Input Parameters	Data Type	Description	Valid and Default Values
		auto tuning of the CV2 output. Ignored when CV2 is not in Manual mode. The function block resets the input to FALSE.	
AtuneCV3Start	BOOL	Start Autotune request for CV3. Set True to initiate auto tuning of the CV3 output. Ignored when CV3 is not in Manual mode. The function block resets the input to FALSE.	Default = FALSE
AtuneCV1UseModel	BOOL	Use Autotune model request for CV1. Set True to replace the current model parameters with the calculated Autotune model parameters. The function block resets the input parameter to FALSE.	Default = FALSE
AtuneCV2UseModel	BOOL	Use Autotune model request for CV2. Set True to replace the current model parameters with the calculated Autotune model parameters. The function block resets the input parameter to FALSE.	Default = FALSE
AtuneCV3UseModel	BOOL	Use Autotune model request for CV3. Set True to replace the current model parameters with the calculated Autotune model parameters. The function block resets the input parameter to FALSE.	Default = FALSE
AtuneCV1Abort	BOOL	Abort Autotune request for CV1. Set True to abort the auto tuning of CV1 output. The function block resets input parameter to FALSE.	Default = FALSE
AtuneCV2Abort	BOOL	Abort Autotune request for CV2. Set True to abort the auto tuning of CV2 output.	Default = FALSE

Input Parameters	Data Type	Description	Valid and Default Values
		The function block resets input parameter to FALSE.	
AtuneCV3Abort	BOOL	Abort Autotune request for CV3. Set True to abort the auto tuning of CV3 output. The function block resets input parameter to FALSE.	Default = FALSE

Output Parameters	Data Type	Description	Valid and Default Values
EnableOut	BOOL	Indicates if instruction is enabled. Cleared to false if CV1EU, CV2EU or CV3EU overflows.	
CV1EU	REAL	Scaled control variable output for CV1. Scaled by using CV1EUMax and CV1EUMin, where CV1EUMax corresponds to 100% and CV1EUMin corresponds to 0%. This output is typically used to control an analog output module or a secondary loop. CV1EU = (CV1 * CV1EUSpan / 100) + CV1EUMin CV1EU span calculation: CV1EUSpan = (CV1EUMax – CV1EUMin)	
CV2EU	REAL	Scaled control variable output for CV2. Scaled by using CV2EUMax and CV2EUMin, where CV2EUMax corresponds to 100% and CV2EUMin corresponds to 0%. This output is typically used to control an analog output module or a secondary loop. CV2EU = (CV2 * CV2EUSpan / 100) + CV2EUMin	

Output Parameters	Data Type	Description	Valid and Default Values
		CV2EU span calculation: CV2EUSpan = (CV2EUMax – CV2EUMin)	
CV3EU	REAL	Scaled control variable output for CV3. Scaled by using CV3EUMax and CV3EUMin, where CV3EUMax corresponds to 100% and CV3EUMin corresponds to 0%. This output is typically used to control an analog output module or a secondary loop. CV3EU = (CV3 * CV3EUSpan / 100) + CV3EUMin CV3EU span calculation: CV3EUSpan = (CV3EUMax – CV3EUMin)	
CV1	REAL	Control variable 1 output. This value will always be expressed as 0...100%. CV1 is limited by CV1HLimit and CV1LLimit when in Auto mode or in Manual mode if CVManLimiting is TRUE; otherwise limited by 0 and 100%.	
CV2	REAL	Control variable 2 output. This value will always be expressed as 0...100%. CV2 is limited by CV2HLimit and CV2LLimit when in Auto mode or in Manual mode if CVManLimiting is TRUE; otherwise limited by 0 and 100%.	
CV3	REAL	Control variable 3 output. This value will always be expressed as 0...100%. CV3 is limited by CV3HLimit and CV3LLimit when in Auto mode or in Manual mode if CVManLimiting is TRUE;	

Output Parameters	Data Type	Description	Valid and Default Values
		otherwise limited by 0 and 100%.	
DeltaCV1	REAL	Difference between the Current CV1 and the previous CV1 (Current CV1 - previous CV1).	
DeltaCV2	REAL	Difference between the Current CV2 and the previous CV2 (Current CV2 - previous CV2).	
DeltaCV3	REAL	Difference between the Current CV3 and the previous CV3 (Current CV3 - previous CV3).	
CV1Initializing	BOOL	Initialization mode indicator for CV1. Set TRUE when CV1InitReq or function blockFirstScan are TRUE, or on a TRUE to FALSE transition of CV1Fault (bad to good). CV1Initializing is set FALSE after the function block has been initialized and CV1InitReq is no longer TRUE.	
CV2nitializing	BOOL	Initialization mode indicator for CV2. Set TRUE when CV2InitReq, function blockFirstScan or OLCFirstRun, are TRUE, or on a TRUE to FALSE transition of CV2Fault (bad to good). CV2Initializing is set FALSE after the function block has been initialized and CV2InitReq is no longer TRUE.	
CV3initializing	BOOL	Set TRUE when CV3InitReq, function blockFirstScan or OLCFirstRun, are TRUE, or on a TRUE to FALSE transition of CV3Fault (bad to good). CV3Initializing is set FALSE after the	

Output Parameters	Data Type	Description	Valid and Default Values
		function block has been initialized and CV3InitReq is no longer TRUE.	
CV1HAlarm	BOOL	CV1 high alarm indicator. TRUE when the calculated value for CV1 > 100 or CV1HLimit.	
CV12HAlarm	BOOL	CV2 high alarm indicator. TRUE when the calculated value for CV2 > 100 or CV2HLimit.	
CV3HAlarm	BOOL	CV3 high alarm indicator. TRUE when the calculated value for CV3 > 100 or CV3HLimit.	
CV1LAlarm	BOOL	CV1 low alarm indicator. TRUE when the calculated value for CV1 < 0 or CV1LLimit.	
CV2LAlarm	BOOL	CV2 low alarm indicator. TRUE when the calculated value for CV2 < 0 or CV2LLimit.	
CV3LAlarm	BOOL	CV3 low alarm indicator. TRUE when the calculated value for CV3 < 0 or CV3LLimit.	
CV1ROCPosAlarm	BOOL	CV1 rate of change alarm indicator. TRUE when the calculated rate of change for CV1 exceeds CV1ROCPosLimit.	
CV2ROCPosAlarm	BOOL	CV2 rate of change alarm indicator. TRUE when the calculated rate of change for CV2 exceeds CV2ROCPosLimit.	
CV3ROCPosAlarm	BOOL	CV3 rate of change alarm indicator. TRUE when the calculated rate of change for CV3 exceeds CV3ROCPosLimit.	

Output Parameters	Data Type	Description	Valid and Default Values
CV1ROCNegAlarm	BOOL	CV1 rate of change alarm indicator. TRUE when the calculated rate of change for CV1 exceeds CV1ROCNegLimit.	
CV2ROCNegAlarm	BOOL	CV2 rate of change alarm indicator. TRUE when the calculated rate of change for CV2 exceeds CV2ROCNegLimit.	
CV3ROCNegAlarm	BOOL	CV3 rate of change alarm indicator. TRUE when the calculated rate of change for CV3 exceeds CV3ROCNegLimit.	
SP	REAL	Current setpoint value. The value of SP is used to control CV when in the Auto or the PV Tracking mode, scaled in PV units.	
SPPercent	REAL	The value of SP expressed in percent of span of PV. SPPercent = ((SP – PVEUMin) * 100) / PVSpan. PV Span calculation: PVSpan = (PVEUMax – PVEUMin)	
SPHAlarm	BOOL	SP high alarm indicator. TRUE when the SP ≥ SPHLimit.	
SPLAlarm	BOOL	SP low alarm indicator. TRUE when the SP ≤ SPLLimit.	
PVPercent	REAL	PV expressed in percent of span. PVPercent = ((PV – PVEUMin) * 100) / PVSpan PV Span calculation: PVSpan = (PVEUMax – PVEUMin)	

Output Parameters	Data Type	Description	Valid and Default Values
E	REAL	Process error. Difference between SP and PV, scaled in PV units.	
EPercent	REAL	The error expressed as a percent of span.	
CV1WindupHOut	BOOL	CV1 Windup high indicator. TRUE when either a SP high or CV1 high/low limit has been reached. This signal will typically be used by the WindupHIn input to limit the windup of the CV1 output on a primary loop.	
CV2WindupHOut	BOOL	CV2 Windup high indicator. TRUE when either a SP high or CV2 high/low limit has been reached. This signal will typically be used by the WindupHIn input to limit the windup of the CV2 output on a primary loop.	
CV3WindupHOut	BOOL	CV3 Windup high indicator. TRUE when either a SP high or CV3 high/low limit has been reached. This signal will typically be used by the WindupHIn input to limit the windup of the CV3 output on a primary loop.	
CV1WindupLOut	BOOL	CV1 Windup low indicator. TRUE when either a SP or CV1 high/low limit has been reached. This signal will typically be used by the WindupLIn input to limit the windup of the CV1 output on a primary loop.	
CV2WindupLOut	BOOL	CV2 Windup low indicator. TRUE when either a SP or CV2 high/low limit has been reached. This signal will typically be used by the WindupLIn input to	

Output Parameters	Data Type	Description	Valid and Default Values
		limit the windup of the CV2 output on a primary loop.	
CV3WindupLOut	BOOL	CV3 Windup low indicator. TRUE when either a SP or CV3 high/low limit has been reached. This signal will typically be used by the WindupLIn input to limit the windup of the CV3 output on a primary loop.	
ProgOper	BOOL	Program/Operator control indicator. TRUE when in Program control. FALSE when in Operator control.	
CV1Auto	BOOL	Auto mode indicator for CV1. TRUE when CV1 in the Auto mode.	
CV2Auto	BOOL	Auto mode indicator for CV2. TRUE when CV2 in the Auto mode.	
CV2Auto	BOOL	Auto mode indicator for CV3. TRUE when CV3 in the Auto mode.	
CV1Manual	BOOL	Manual mode indicator CV1. TRUE when CV1 in the Manual mode.	
CV2Manual	BOOL	Manual mode indicator CV2. TRUE when CV2 in the Manual mode.	
CV3Manual	BOOL	Manual mode indicator CV3. TRUE when CV3 in the Manual mode.	
CV1Override	BOOL	Override mode indicator for CV1. TRUE when CV1 in the Override mode.	
CV2Override	BOOL	Override mode indicator for CV2. TRUE when CV2 in the Override mode.	

Output Parameters	Data Type	Description	Valid and Default Values
CV3Override	BOOL	Override mode indicator for CV3. TRUE when CV3 in the Override mode.	
CV1Hand	BOOL	Hand mode indicator for CV1. TRUE when CV1 in the Hand mode.	
CV2Hand	BOOL	Hand mode indicator for CV2. TRUE when CV2 in the Hand mode.	
CV3Hand	BOOL	Hand mode indicator for CV3. TRUE when CV3 in the Hand mode.	
DeltaT	REAL	Elapsed time between updates in seconds.	
CV1StepSizeUsed	REAL	Actual CV1 step size used for tuning.	
CV2StepSizeUsed	REAL	Actual CV2 step size used for tuning.	
CV3StepSizeUsed	REAL	Actual CV3 step size used for tuning.	
CV1GainTuned	REAL	The calculated value of the internal model gain for CV1 after tuning is completed.	
CV2GainTuned	REAL	The calculated value of the internal model gain for CV2 after tuning is completed.	
CV3GainTuned	REAL	The calculated value of the internal model gain for CV3 after tuning is completed.	
CV1TCTuned	REAL	The calculated value of the internal model time constant for CV1 after tuning is completed.	
CV2TCTuned	REAL	The calculated value of the internal model time constant for CV2 after tuning is completed.	
CV3TCTuned	REAL	The calculated value of the internal model time	

Output Parameters	Data Type	Description	Valid and Default Values
		constant for CV3 after tuning is completed.	
CV1DTTuned	REAL	The calculated value of the internal model deadtime for CV1 after tuning is completed.	
CV2DTTuned	REAL	The calculated value of the internal model deadtime for CV2 after tuning is completed.	
CV3DTTuned	REAL	The calculated value of the internal model deadtime for CV3 after tuning is completed.	
CV1RespTCTunedS	REAL	The calculated value of the control variable time constant in slow response speed for CV1 after tuning is completed.	
CV2RespTCTunedS	REAL	The calculated value of the control variable time constant in slow response speed for CV2 after tuning is completed.	
CV3RespTCTunedS	REAL	The calculated value of the control variable time constant in slow response speed for CV3 after tuning is completed.	
CV1RespTCTunedM	REAL	The calculated value of the control variable time constant in medium response speed for CV1 after tuning is completed.	
CV2RespTCTunedM	REAL	The calculated value of the control variable time constant in medium response speed for CV2 after tuning is completed.	
CV3RespTCTunedM	REAL	The calculated value of the control variable time constant in medium	

Output Parameters	Data Type	Description	Valid and Default Values
		response speed for CV3 after tuning is completed.	
CV1RespTCTunedF	REAL	The calculated value of the control variable time constant in fast response speed for CV1 after tuning is completed.	
CV2RespTCTunedF	REAL	The calculated value of the control variable time constant in fast response speed for CV2 after tuning is completed.	
CV3RespTCTunedF	REAL	The calculated value of the control variable time constant in fast response speed for CV3 after tuning is completed.	
AtuneCV1On	BOOL	Set True when auto tuning for CV1 has been initiated.	
AtuneCV2On	BOOL	Set True when auto tuning for CV2 has been initiated.	
AtuneCV3On	BOOL	Set True when auto tuning for CV3 has been initiated.	
AtuneCV1Done	BOOL	Set True when auto tuning for CV1 has completed successfully.	
AtuneCV2Done	BOOL	Set True when auto tuning for CV2 has completed successfully.	
AtuneCV3Done	BOOL	Set True when auto tuning for CV3 has completed successfully.	
AtuneCV1Aborted	BOOL	Set True when auto tuning for CV1 has been aborted by user or due to errors that occurred during the auto tuning operation.	
AtuneCV2Aborted	BOOL	Set True when auto tuning for CV2 has been aborted by user or due to errors	

Output Parameters	Data Type	Description	Valid and Default Values
		that occurred during the auto tuning operation.	
AtuneCV3Aborted	BOOL	Set True when auto tuning for CV3 has been aborted by user or due to errors that occurred during the auto tuning operation.	
AtuneCV1Status	DINT	Indicates the tuning status for CV1.	
AtuneCV2Status	DINT	Indicates the tuning status for CV2.	
AtuneCV3Status	DINT	Indicates the tuning status for CV3.	
AtuneCV1Fault	BOOL	CV1 Autotune has generated any of the following faults.	
AtuneCV2Fault	BOOL	CV2 Autotune has generated any of the following faults.	
AtuneCV3Fault	BOOL	CV3 Autotune has generated any of the following faults.	
AtuneCV1PVOutOfLimit	BOOL	Either PV or the deadtime-step ahead prediction of PV exceeds PVTuneLimit during CV1 Autotuning. When True, CV1 Autotuning is aborted.	
AtuneCV2PVOutOfLimit	BOOL	Either PV or the deadtime-step ahead prediction of PV exceeds PVTuneLimit during CV2 Autotuning. When True, CV2 Autotuning is aborted.	
AtuneCV3PVOutOfLimit	BOOL	Either PV or the deadtime-step ahead prediction of PV exceeds PVTuneLimit during CV3 Autotuning. When True, CV3 Autotuning is aborted.	

Output Parameters	Data Type	Description	Valid and Default Values
AtuneCV1ModeInv	BOOL	The CC mode was not Manual at start of Autotuning or the CC mode was changed from Manual during CV1 Autotuning. When True, CV1 Autotuning is not started or is aborted.	
AtuneCV2ModeInv	BOOL	The CC mode was not Manual at start of Autotuning or the CC mode was changed from Manual during CV2 Autotuning. When True, CV2 Autotuning is not started or is aborted.	
AtuneCV3ModeInv	BOOL	The CC mode was not Manual at start of Autotuning or the CC mode was changed from Manual during CV3 Autotuning. When True, CV3 Autotuning is not started or is aborted.	
AtuneCV1WindupFault	BOOL	CV1WindupHIn or CV1WindupLIn is True at start of CV1 Autotuning or during CV1 Autotuning. When True, CV1 Autotuning is not started or is aborted.	
AtuneCV2WindupFault	BOOL	CV2WindupHIn or CV2WindupLIn is True at start of CV1 Autotuning or during CV2 Autotuning. When True, CV2 Autotuning is not started or is aborted.	
AtuneCV3WindupFault	BOOL	CV3WindupHIn or CV3WindupLIn is True at start of CV3 Autotuning or during CV3 Autotuning. When True, CV3 Autotuning is not started or is aborted.	
AtuneCV1StepSize0	BOOL	CV1StepSizeUsed = 0 at start of CV1 Autotuning. When True, CV1 Autotuning is not started.	
AtuneCV2StepSize0	BOOL	CV2StepSizeUsed = 0 at start of CV2 Autotuning.	

Output Parameters	Data Type	Description	Valid and Default Values
		When True, CV2 Autotuning is not started.	
AtuneCV3StepSize0	BOOL	CV3StepSizeUsed = 0 at start of CV3 Autotuning. When True, CV3 Autotuning is not started.	
AtuneCV1LimitsFault	BOOL	CV1LimitsInv and CVManLimiting are True at start of CV1 Autotuning or during CV1 Autotuning. When True, CV1 Autotuning is not started or is aborted.	
AtuneCV2LimitsFault	BOOL	CV2LimitsInv and CVManLimiting are True at start of CV2 Autotuning or during CV2 Autotuning. When True, CV2 Autotuning is not started or is aborted.	
AtuneCV3LimitsFault	BOOL	CV3LimitsInv and CVManLimiting are True at start of CV3 Autotuning or during CV3 Autotuning. When True, CV3 Autotuning is not started or is aborted.	
AtuneCV1InitFault	BOOL	CV1Initializing is True at start of CV1 Autotuning or during CV1 Autotuning. When True, CV1 Autotuning is not started or is aborted.	
AtuneCV2InitFault	BOOL	CV2Initializing is True at start of CV2 Autotuning or during CV2 Autotuning. When True, CV2 Autotuning is not started or is aborted.	
AtuneCV3InitFault	BOOL	CV3Initializing is True at start of CV3 Autotuning or during CV3 Autotuning. When True, CV3 Autotuning is not started or is aborted.	
AtuneCV1EUSpanChanged	BOOL	CV1EUSpan or PVEUSpan changes during CV1 Autotuning. When True, CV1 Autotuning is aborted.	

Output Parameters	Data Type	Description	Valid and Default Values
AtuneCV2EUSpanChanged	BOOL	CV2EUSpan or PVEUSpan changes during CV2 Autotuning. When True, CV2 Autotuning is aborted.	
AtuneCV3EUSpanChanged	BOOL	CV3EUSpan or PVEUSpan changes during CV3 Autotuning. When True, CV3 Autotuning is aborted.	
AtuneCV1Changed	BOOL	CV1Oper is changed when in Operation control or CV1Prog is changed when in Program control or CV1 becomes high/low or ROC limited during CV1 Autotuning. When True, CV1 Autotuning is aborted.	
AtuneCV2Changed	BOOL	CV2Oper is changed when in Operation control or CV2Prog is changed when in Program control or CV2 becomes high/low or ROC limited during CV2 Autotuning. When True, CV2 Autotuning is aborted.	
AtuneCV3Changed	BOOL	CV3Oper is changed when in Operation control or CV3Prog is changed when in Program control or CV3 becomes high/low or ROC limited during CV3 Autotuning. When True, CV3 Autotuning is aborted.	
AtuneCV1Timeout	BOOL	Elapsed time is greater then AtuneTimeLimit since step test is started. When True, CV1 Autotuning is aborted.	
AtuneCV2Timeout	BOOL	Elapsed time is greater then AtuneTimeLimit since step test is started. When True, CV2 Autotuning is aborted.	
AtuneCV3Timeout	BOOL	Elapsed time is greater then AtuneTimeLimit since step test is started. When	

Output Parameters	Data Type	Description	Valid and Default Values
		True, CV3 Autotuning is aborted.	
AtuneCV1PVNotSettled	BOOL	The PV is changed too much to Autotune for CV1. When True, CV1 Autotuning is aborted. Wait until PV is more stable before autotuning CV1.	
AtuneCV2PVNotSettled	BOOL	The PV is changed too much to Autotune for CV2. When True, CV2 Autotuning is aborted. Wait until PV is more stable before autotuning CV2.	
AtuneCV3PVNotSettled	BOOL	The PV is changed too much to Autotune for CV3. When True, CV3 Autotuning is aborted. Wait until PV is more stable before autotuning CV3.	
Status1	DINT	Bit mapped status of the function block.	
Status2	DINT	Additional bit mapped status for the function block.	
Status3CV1	DINT	Additional bit mapped CV1 status for the function block. A value of 0 indicates that no faults have occurred.	
Status3CV2	DINT	Additional bit mapped CV2 status for the function block. A value of 0 indicates that no faults have occurred.	
Status3CV3	DINT	Additional bit mapped CV3 status for the function block. A value of 0 indicates that no faults have occurred.	
InstructFault	BOOL	The function block has generated a fault. Indicates state of bits	

Output Parameters	Data Type	Description	Valid and Default Values
		in Status1, Status2, and Status3CV(n), where (n) can be 1, 2, or 3. A value of 0 indicates that no faults have occurred. Any parameters that could be configured with an invalid value must have a status parameter to indicate their invalid status.	
PVFaulted	BOOL	Process variable PV health bad.	
PVSpanInv	BOOL	The span of PV inValid, PVEUMax < PVEUMin.	
SPProgInv	BOOL	SPProg < SPLLimit or > SPHLimit. Limit value used for SP.	
SPOperInv	BOOL	SPOper < SPLLimit or > SPHLimit. Limit value used for SP.	
SPLimitsInv	BOOL	Limits inValid: SPLLimit < PVEUMin, SPHLimit > PVEUMax, or SPHLimit < SPLLimit. If SPHLimit < SPLLimit, then limit value by using SPLLimit.	
SampleTimeTooSmall	BOOL	Model DeadTime / DeltaT must be less than or equal to 200.	
FactorInv	BOOL	Entered value for Factor < 0.	
TimingModeInv	BOOL	Entered TimingMode inValid. If the current mode is not Override or Hand then set to Manual mode.	
RTSMissed	BOOL	Only used when in Real Time Sampling mode. Is TRUE when ABS(DeltaT – RTSTime) > 1 millisecond.	

Output Parameters	Data Type	Description	Valid and Default Values
RTSTimeInv	BOOL	Entered RTSTime inValid.	
RTSTimeStampInv	BOOL	RTSTimeStamp inValid. If the current mode is not Override or Hand, then set to Manual mode.	
DeltaTInv	BOOL	DeltaT inValid. If the current mode is not Override or Hand then, set to Manual mode.	
CV1Faulted	BOOL	Control variable CV1 health bad.	
CV2Faulted	BOOL	Control variable CV2 health bad.	
CV3Faulted	BOOL	Control variable CV3 health bad.	
CV1HandFBFaulted	BOOL	CV1 HandFB value health bad.	
CV2HandFBFaulted	BOOL	CV2 HandFB value health bad.	
CV3HandFBFaulted	BOOL	CV3 HandFB value health bad.	
CV1ProgInv	BOOL	CV1Prog < 0 or > 100, or < CV1LLimit or > CV1HLimit when CVManLimiting is TRUE. Limit value used for CV1.	
CV2ProgInv	BOOL	CV2Prog < 0 or > 100, or < CV2LLimit or > CV2HLimit when CVManLimiting is TRUE. Limit value used for CV2.	
CV3ProgInv	BOOL	CV3Prog < 0 or > 100, or < CV3LLimit or > CV3HLimit when CVManLimiting is TRUE. Limit value used for CV3.	
CV1OperInv	BOOL	CV1Oper < 0 or > 100, or < CV1LLimit or > CV1HLimit when CVManLimiting is	

Output Parameters	Data Type	Description	Valid and Default Values
		TRUE. Limit value used for CV1.	
CV2OperInv	BOOL	CV2Oper < 0 or > 100, or < CV2LLimit or> CV2HLimit when CVManLimiting is TRUE. Limit value used for CV2.	
CV3OperInv	BOOL	CV3Oper < 0 or > 100, or < CV3LLimit or> CV3HLimit when CVManLimiting is TRUE. Limit value used for CV3.	
CV1OverrideValueInv	BOOL	CV1OverrideValue < 0 or > 100. Limit value used for CV1.	
CV2OverrideValueInv	BOOL	CV2OverrideValue < 0 or > 100. Limit value used for CV2.	
CV3OverrideValueInv	BOOL	CV3OverrideValue < 0 or > 100. Limit value used for CV3.	
CV1TrackValueInv	BOOL	Entered CV1TrackValue < 0 or > 100. Limit value used for CV1.	
CV2TrackValueInv	BOOL	Entered CV2TrackValue < 0 or > 100. Limit value used for CV2.	
CV3TrackValueInv	BOOL	Entered CV3TrackValue < 0 or > 100. Limit value used for CV3.	
CV1EUSpanInv	BOOL	The span of CV1EU inValid, CV1EUMax equals CV1EUMin.	
CV2EUSpanInv	BOOL	The span of CV2EU inValid, CV2EUMax equals CV2EUMin.	
CV3EUSpanInv	BOOL	The span of CV3EU inValid, CV3EUMax equals CV3EUMin.	

Output Parameters	Data Type	Description	Valid and Default Values
CV1LimitsInv	BOOL	CV1LLimit < 0, CV1HLimit > 100, or CV1HLimit <= CV1LLimit. If CV1HLimit <= CV1LLimit, limit CV1 by using CV1LLimit.	
CV2LimitsInv	BOOL	CV2LLimit < 0, CV2HLimit > 100, or CV2HLimit <= CV2LLimit. If CV2HLimit <= CV2LLimit, limit CV2 by using CV2LLimit.	
CV3LimitsInv	BOOL	CV3LLimit < 0, CV3HLimit > 100, or CV3HLimit <= CV3LLimit. If CV3HLimit <= CV3LLimit, limit CV3 by using CV3LLimit.	
CV1ROCLimitInv	BOOL	CV1ROCLimit < 0, disables CV1 ROC limiting.	
CV2ROCLimitInv	BOOL	CV2ROCLimit < 0, disables CV2 ROC limiting.	
CV3ROCLimitInv	BOOL	CV3ROCLimit < 0, disables CV3 ROC limiting.	
CV1HandFBInv	BOOL	CV1HandFB < 0 or > 100. Limit value used for CV1.	
CV2HandFBInv	BOOL	CV2HandFB < 0 or > 100. Limit value used for CV2.	
CV3HandFBInv	BOOL	CV3HandFB < 0 or > 100. Limit value used for CV3.	
CV1ModelGainInv	BOOL	CV1ModelGain is 1.#QNAN or -1.#IND (Not A Number), or ± 1.$ (Infinity ∞).	
CV2ModelGainInv	BOOL	CV2ModelGain is 1.#QNAN or -1.#IND (Not A Number), or ± 1.$ (Infinity ∞).	
CV3ModelGainInv	BOOL	CV3ModelGain is 1.#QNAN or -1.#IND (Not A Number), or ± 1.$ (Infinity ∞).	
CV1ModelTCInv	BOOL	CV1ModelTC < 0.	
CV2ModelTCInv	BOOL	CV2ModelTC < 0.	

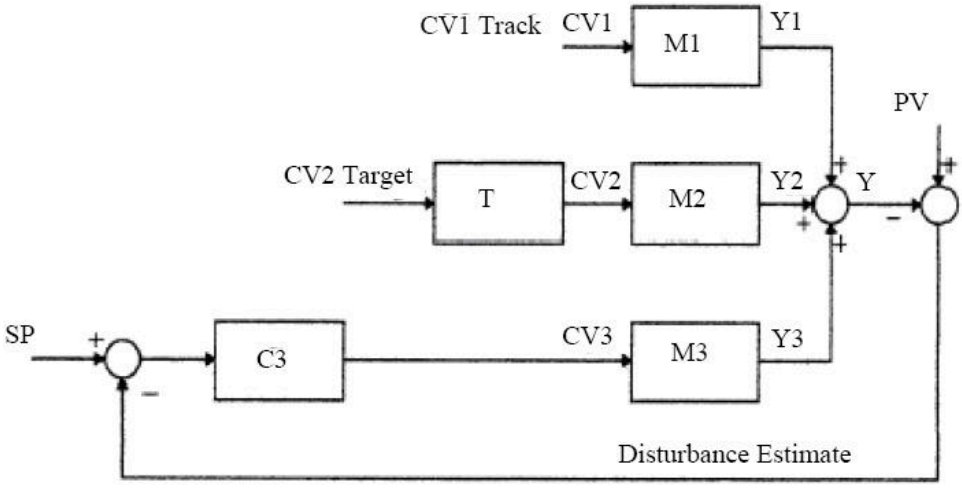
Output Parameters	Data Type	Description	Valid and Default Values
CV3ModelTCInv	BOOL	CV3ModelTC < 0.	
CV1ModelDTInv	BOOL	CV1ModelDT < 0.	
CV2ModelDTInv	BOOL	CV2ModelDT < 0.	
CV3ModelDTInv	BOOL	CV3ModelDT < 0.	
CV1RespTCInv	BOOL	CV1RespTC < 0.	
CV2RespTCInv	BOOL	CV2RespTC < 0.	
CV3RespTCInv	BOOL	CV3RespTC < 0.	
CV1TargetInv	BOOL	CV1Target < 0. or > 100.	
CV2TargetInv	BOOL	CV2Target < 0. or > 100.	
CV3TargetInv	BOOL	CV3Target < 0. or > 100.	

Description

Coordinated Control is a flexible model-based algorithm that can be used in various configurations, for example:

- Three control variables are used to control one process variable
- Heat-cool split range control
- Feedforward control
- Zone temperature control
- Constraint control

The following illustration is an example of the Coordinated Control closed loop configuration.



In this example, CV1 is in Manual mode, CV2 is driven to its target value, and CV3 is the active control. The following table describes this example in detail.

Name	Description
CV1	Is in Manual mode
CV2	Is driven to its target value (CV2 = Target1stCV)
CV3	Is the active control (CV3 = Act1stCV)

This example could be a heat cooled system with a feed forward where:

- CV1 is feed forward;
- CV2 is cooling;
- CV3 heating.

Since CV1 is in Manual mode, CV3 target value as the lowest priority goal cannot be accomplished. PV will be maintained at the setpoint by using CV3, and at the same time CV2 will be driven to its target value (2nd priority goal).

If the operator changes the CV1 manual value, the control variable will take the change into account when calculating new CV3 and CV2.

M1	CV1 - PV First order lag with deadtime model
M2	CV2 - PV First order lag with deadtime model
M3	CV3 - PV First order lag with deadtime model
T	Target Response
C3	Model based algorithm to control PV by using CV3
Y1, Y2, Y3	Model outputs of M1, M2, M3
Y	PV prediction

Affects Math Status Flags

No

Major/Minor Faults

None specific to this instruction. See Common Attributes on page for operand-related faults.

Execution

Function Block

Condition/State	Action Taken
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false.

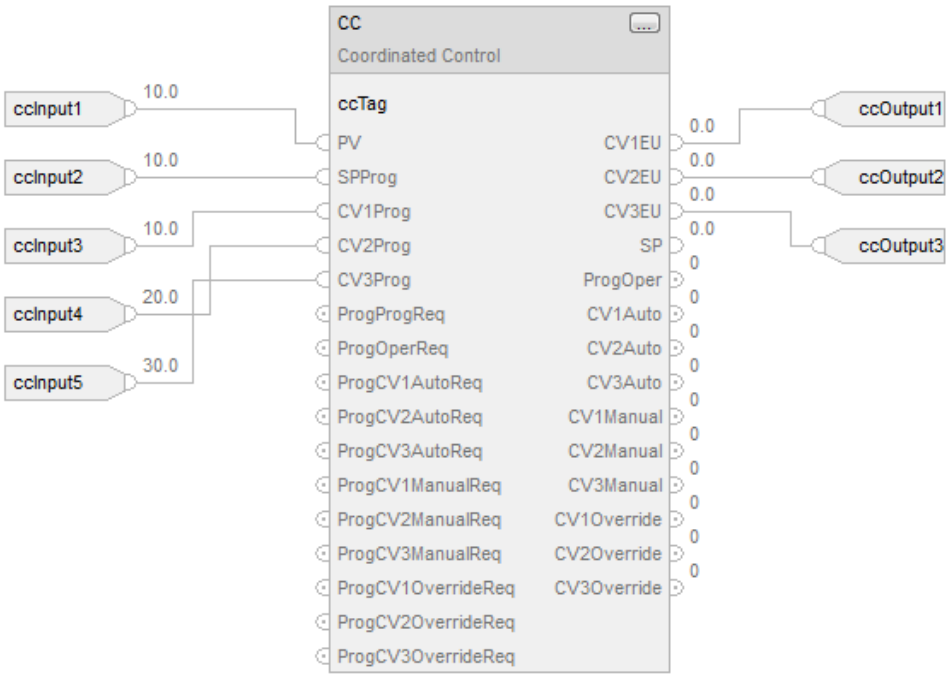
Condition/State	Action Taken
Tag.EnableIn is true	EnableIn and EnableOut bits are set to true. The instruction executes.
Instruction first run	N/A.
Instruction first scan	N/A
Postscan	EnableIn and EnableOut bits are cleared to false.

Structured Text

Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.
Normal Execution	See Tag.EnableIn is true in the Function Block table.
Postscan	See Postscan in the Function Block table.

Example

Function Block



Structured Text

```
ccTag.PV := ccInput1;
ccTag.SPProg := ccInput2;
ccTag.CV1Prog := ccInput3;
```

```
ccTag.CV2Prog := ccInput4;
ccTag.CV3Prog := ccInput5;
CC(ccTag);
ccOutput1 := ccTag.CV1EU;
ccOutput2 := ccTag.CV2EU;
ccOutput3 := ccTag.CV3EU;
```

CC Function Block Configuration

Starting with the default configuration, configure the following parameters:

Parameter	Description
PVEUMax	Maximum scaled value for PV.
PVEUMin	Minimum scaled value for PV.
SPHLimit	SP high limit value, scaled in PV units.
PPLLimit	SP low limit value, scaled in PV units.
CV1InitValue	An initial value of the control variable CV1 output.
CV2InitValue	An initial value of the control variable CV2 output.
CV3InitValue	An initial value of the control variable CV3 output.

If you have the process models available, you can intuitively tune the CC control variable by entering the following parameters:

Parameter	Description
ModelGains	Nonzero numbers (negative for direct acting control variable, positive for reverse acting control variable)
ModelTimeConstants	Always positive numbers
ModelDeadtimes	Always positive numbers
ResponseTimeConstants	Always positive numbers
Active 1st, 2nd and 3rd CV	Specify the order in which CV's will be used to compensate for PV - SP error.
Target 1st, 2nd and 3rd CV	Specify the priorities in which CV's will be driven to their respective target values.
CVTargetValues	Specify to which values should the control variable drive the individual CV's
TargetRespTC	Specify the speed of CV's to approach the target values

The function block behaves in a defined way for any combination of CV Active and Target lists and CV Auto-Manual modes. The function block attempts to accomplish these goals in the following order of priorities:

1. Control PV to SP
2. Control Target1stCV to its target value
3. Control Target2ndCV to its target value

If any CV is put in Manual mode, the CC function block gives up the goal with priority 3. If two CV's are in Manual mode, the CC function block is reduced to an IMC, (single input, single output) control variable controlling the PV to its setpoint.

In addition to this, however, the control variable reads the Manual CV values from the CV's that are in Manual mode as feedforward signals. Then, the CC function block predicts the influence of the Manual CV values on the PV by using the appropriate internal models, and calculates the third CV that remains in Auto mode.

For integrating process types (such as level control and position control), internal nonintegrating models are used to approximate the integrating process. The Factor parameter is used to convert the identified integrating process models to nonintegrating internal models used for CV calculation. This is necessary to provide for stable function block execution.

CC Function Block Model Initialize

A model initialization occurs:

- During First Scan of the block
- When the ModelInit request parameter is set
- When DeltaT changes

You may need to manually adjust the internal model parameters or the response time constants. You can do so by changing the appropriate parameters and setting the appropriate ModelInit bit. The internal states of the control variable will be initialized, and the bit will automatically reset.

For example, modify the Model Gain for CV2 - PV model. Set the ModelInit2 parameter to TRUE to initialize the CV2 - PV internal model parameters and for the new model gain to take effect.

CC Function Block Tuning

The function block is equipped with an internal tuner (modeler). The purpose of the tuner is to identify the process model parameters and to use these parameters as internal model parameters (gain, time constant, and deadtime). The tuner also calculates an optimal response time constant. Set the tuner by configuring the following parameters for each CV - PV process.

ProcessType	Integral (level, position control) or nonintegrating (flow, pressure control)
ProcessGainSign	Set to indicate a negative process gain (increase in output causes a decrease in PV); reset to indicate a positive process gain (increase in output causes an increase in PV).
ResponseSpeed	Slow, medium, or fast, based on control objective.
NoiseLevel	An estimate of noise level on PV-low, medium, or high-such that the tuner can distinguish which PV

	change is a random noise and which is caused by the CV step change.
StepSize	A nonzero positive or negative number defining the magnitude of CV step change in either positive or negative direction, respectively.
PVTuneLimit	(only for integrating process type) in PV engineering units, defines how much of PV change that is caused by CV change to tolerate before aborting the tuning test due to exceeding this limit.

The tuner is started by setting the appropriate AtuneStart bit (AtuneCV1Start, for example). You can stop the tuning by setting the appropriate AtuneAbort bit.

After the tuning is completed successfully, the GainTuned, TCTuned, DTTuned, and RespTCTuned parameters are updated with the tuning results, and the AtuneStatus code is set to indicate complete.

You can copy these parameters to the ModelGain, ModelTC, and ResponseTC, respectively, by setting the AtuneUseModel bit. The control variable will automatically initialize the internal variables and continue normal operation. It will automatically reset the AtuneUseModel bit.

CC Function Block Tuning Errors

If an error occurs during the tuning procedure, the tuning is aborted, and an appropriate AtuneStatus value is set. Also, a user can abort the tuning by setting the AtuneAbort parameter.

After an abort, the CV will assume its value before the step change, and the GainTuned, TCTuned, DTTuned, and RespTCTuned parameters are not updated. The AtuneStatus parameter identifies the reason for the abort.

Configure CC Function Block tuner

Follow these steps to configure the tuner.

To configure the CC Function Block tuner

1. Put all three CV parameters into Manual mode.
2. Set the AtuneStart parameter. The tuner starts collecting PV and CV data for noise calculation.
3. After collecting 60 samples (60*DeltaT) period, the tuner adds StepSize to the CV.
4. Set the AtuneUseModel parameter to copy the tuned parameters to the model parameters. The function block then resets the AtuneUseModel parameter.

- After successfully collecting the PV data as a result of the CV step change, the CV assumes its value before the step change and the AtuneStatus, GainTuned, TCTuned, DTTuned, and RespTCTuned parameters are updated.
- After a successful AutoTuneDone, the Atune parameter is set to one (1). Tuning completed successfully.
- To identify models and to calculate response time constants for all three CV-PV processes, run the tuner up to three times to obtain CV1-PV, CV2-PV, and CV3-PV models and tuning, respectively.

Internal Model Control (IMC)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, Compact GuardLogix 5380, CompactLogix 5380, ControlLogix 5580, GuardLogix 5580, and ControlLogix 5590 controllers.

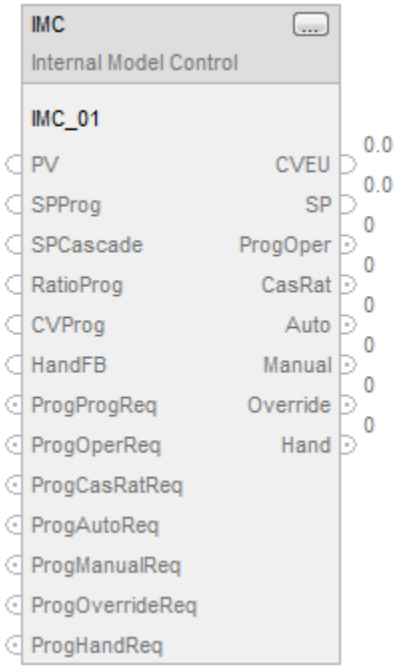
The Internal Model Control (IMC) instruction controls a single process variable by manipulating a single control-variable output. This function block performs an algorithm where the actual error signal is compared against that of an internal first-order lag plus deadtime model of the process. The IMC function block calculates the control variable output (CV) in the Auto mode based on the PV - SP deviation, internal model, and tuning.

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram.

Function Block



Structured Text

IMC(IMC_tag);

Operands

Function Block

Operands:	Type:	Format	Description:
IMC tag	INTERNAL MODEL CONTROL	Structure	IMC Structure

Structured Text

Operands:	Type:	Format	Description:
IMC tag	INTERNAL MODEL CONTROL	Structure	IMC Structure

See *Structured Text Syntax* for more information on the syntax of expressions within structured text.

IMPORTANT: Whenever an APC block detects a change in Delta Time (DeltaT), a ModelInit will be performed. For this reason the blocks should only be run in one of the TimingModes in which DeltaT will be constant.

- TimingMode = 0 (Periodic) while executing these function blocks in a Periodic Task
- TimingMode = 1 (Oversample)

In either case, if the Periodic Task time is dynamically changed, or the OversampleDT is dynamically changed, the block will perform a ModelInit.

The following TimingMode setting are not recommended due to jitter in DeltaT:

- TimingMode = 0 (Periodic) while executing these function blocks in a Continuous or Event Task
- TimingMode = 2 (RealTimeSample)

Structure

Input Parameters	Data Type	Description	Valid and Default Values
EnableIn	BOOL	Enable Input. If False, the function block will not execute and outputs are not updated.	Default=TRUE
PV	REAL	Scaled process variable input. This value is typically read from an analog input module.	Valid = any float Default = 0.0
PVFault	BOOL	PV bad health indicator. If PV is read from an analog input, then PVFault will normally be controlled by the analog input fault status. If PVFault is TRUE, it indicates an error on the input module, set bit in Status.	Default = FALSE FALSE = Good Health
PVUEMax	REAL	Maximum scaled value for PV. The value of PV and SP that corresponds to 100% span of the Process Variable. If PVEUMax ≤ PVEUMin, set bit in Status.	Valid = PVEUMin < PVEUMax ≤ maximum positive float Default = 100.0

Input Parameters	Data Type	Description	Valid and Default Values
PVUEMin	REAL	Minimum scaled value for PV. The value of PV and SP that corresponds to 0% span of the Process Variable. If PVEUMax ≤ PVEUMin, set bit in Status.	Valid = maximum negative float ≤ PVEUMin < PVEUMax Default = 0.0
SPProg	REAL	SP Program value, scaled in PV units. SP is set to this value when in Program control. If value of SPProg or SPOper < SPLLimit or > SPHLimit, set bit in Status and limit value used for SP.	Valid = SPLLimit to SPHLimit Default = 0.0
SPOper	REAL	SP Operator value, scaled in PV units. SP set to this value when in Operator control. If value of SPProg or SPOper < SPLLimit or > SPHLimit, set bit in Status and limit value used for SP.	Valid = SPLLimit to SPHLimit Default = 0.0
SPCascade	REAL	SP Cascade value, scaled in PV units. If CascadeRatio mode and UseRatio is FALSE, then SP is set to this value, typically this will be CVEU of a primary loop. If CascadeRatio mode and UseRatio is TRUE, then SP is set to this value times Ratio. If value of SPCascade < SPLLimit or > SPHLimit, set bit in Status and limit value used for SP.	Valid = SPLLimit to SPHLimit Default = 0.0
SPHLimit	REAL	SP high limit value, scaled in PV units. If SPHLimit < SPLLimit or SPHLimit > PVEUMax, set bit in Status.	Valid = SPLLimit to PVEUMax Default = 100.0
SPLLimit	REAL	SP low limit value, scaled in PV units.	Valid = PVEUMin to SPHLimit

Input Parameters	Data Type	Description	Valid and Default Values
		If SPLLimit < PVEUMin, or SPHLimit < SPLLimit, set bit in Status and limit SP by using the value of SPLLimit.	Default = 0.0
UseRatio	BOOL	Allow Ratio control permissive. Set TRUE to enable ratio control when in CascadeRatio mode.	Default = FALSE
RatioProg	REAL	Ratio Program multiplier, no units (for example, scalar). Ratio and RatioOper are set to this value when in Program control. If RatioProg or RatioOper < RatioLLimit or > RatioHLimit, set bit in Status and limit value used for Ratio.	Valid = RatioLLimit to RatioHLimit Default = 1.0
RatioOper	REAL	Ratio Operator multiplier, no units (for example, scalar). Ratio is set to this value when in Operator control. If RatioProg or RatioOper < RatioLLimit or > RatioHLimit, set bit in Status and limit value used for Ratio.	Valid = RatioLLimit to RatioHLimit Default = 1.0
RatioHLimit	REAL	Ratio high limit value, no units (for example, scalar). Limits the value of Ratio obtained from RatioProg or RatioOper. If RatioLLimit < 0, set bit in Status and limit to zero. If RatioHLimit < RatioLLimit, set bit in Status and limit Ratio by using the value of RatioLLimit.	Valid = RatioLLimit to maximum positive float Default = 1.0
RatioLLimit	REAL	Ratio low limit value, no units (for example, scalar). Limits the value of Ratio	Valid = 0.0 to RatioHLimit Default = 1.0

Input Parameters	Data Type	Description	Valid and Default Values
		obtained from RatioProg or RatioOper. If RatioLLimit < 0, set bit in Status and limit to zero. If RatioHLimit < RatioLLimit, set bit in Status and limit Ratio by using the value of RatioLLimit.	
CVFault	BOOL	Control variable bad health indicator. If CVEU controls an analog output, then CVFault will normally come from the analog output's fault status. If CVFault is TRUE, it indicates an error on the output module, set bit in Status.	Default = FALSE FALSE = Good Health
CVInitReq	BOOL	CV initialization request. While TRUE, set CVEU to the value of CVInitValue. This signal will normally be controlled by the In Hold status on the analog output module controlled by CVEU or from the InitPrimary output of a secondary IMC loop.	Default = FALSE
CVInitValue	REAL	CVEU initialization value, scaled in CVEU units. When CVInitializing is TRUE set CVEU equal to CVInitValue and CV to the corresponding percentage value. CVInitValue will normally come from the feedback of the analog output controlled by CVEU or from the setpoint of a secondary loop. The function block initialization is disabled when CVFaulted or CVEUSpanInv are TRUE (bad).	Valid = any float Default = 0.0
CVProg	REAL	CV Program-Manual value. CV is set to this value	Valid = 0.0 to 100.0 Default = 0.0

Input Parameters	Data Type	Description	Valid and Default Values
		when in Program control and Manual mode. If value of CVProg or CVOper < 0 or > 100, or < CVLLimit or > CVHLimit when CVManLimiting is TRUE, set unique Status bit and limit value used for CV.	
CVOper	REAL	CV Operator-Manual value. CV is set to this value when in Operator control and Manual mode. If not Operator-Manual mode, set CVOper to the value of CV at the end of each function block execution. If value of CVProg or CVOper < 0 or > 100, or < CVLLimit or > CVHLimit when CVManLimiting is TRUE, set unique Status bit and limit value used for CV.	Valid = 0.0 to 100.0 Default = 0.0
CVOverrideValue	REAL	CV Override value. CV set to this value when in Override mode. This value should correspond to a safe state output of the IMC loop. If value of CVOverrideValue < 0 or >100, set unique Status bit and limit value used for CV.	Valid = 0.0 to 100.0 Default = 0.0
CVTrackValue	REAL	CV track value. When CVTrackReq is enabled and the IMC function block is in Manual, the CVTrackValue will be ignored, and the IMC internal model will update its historical data with the CVOper or CVProg value. When CVTrackReq is enabled and the IMC function block is in Auto, the internal model will update its historical data based on the value of CVTrackValue. The CV in	Valid = 0.0 to 100.0 Default = 0.0

Input Parameters	Data Type	Description	Valid and Default Values
		this case will be allowed to move as if the IMC function block was still controlling the process. This is useful in multiloop selection schemes where you want the IMC function block to follow the output of a different controlling algorithm, where you would connect the output of the controlling algorithm into the CVTrackValue.	
CVManLimiting	BOOL	Limit CV in Manual mode request. If Manual mode and CVManLimiting is TRUE, CV will be limited by the CVHLimit and CVLLimit values.	Default = FALSE
CVEUMax	REAL	Maximum value for CVEU. The value of CVEU that corresponds to 100% CV. If CVEUMax = CVEUMin, set bit in Status.	Valid = any float Default = 100.0
CVEUMin	REAL	Minimum value of CVEU. The value of CVEU that corresponds to 0% CV. If CVEUMax = CVEUMin, set bit in Status.	Valid = any float Default = 0.0
CVHLimit	REAL	CV high limit value. This is used to set the CVHAlarm output. It is also used for limiting CV when in Auto or CascadeRatio modes or Manual mode if CVManLimiting is TRUE. If CVLLimit < 0, if CVHLimit > 100, if CVHLimit < CVLLimit, set bit in Status. If CVHLimit < CVLLimit, limit CV by using the value of CVLLimit.	Valid = CVLLimit < CVHLimit ≤ 100.0 Default = 100.0
CVLLimit	REAL	CV low limit value. This is used to set the CVLAlarm output. It is also used for limiting CV when in	Valid = 0.0 ≤ CVLLimit < CVHLimit Default = 0.0

Input Parameters	Data Type	Description	Valid and Default Values
		Auto or CascadeRatio modes or Manual mode if CVManLimiting is TRUE. If CVLLimit < 0, if CVHLimit > 100, if CVHLimit < CVLLimit, set bit in Status. If CVHLimit < CVLLimit, limit CV by using the value of CVLLimit.	
CVROCPosLimit	REAL	CV increasing rate of change limit, in percent per second. Rate of change limiting is only used when in Auto or CascadeRatio modes or Manual mode if CVManLimiting is TRUE. A value of zero disables CV ROC limiting. If value of CVROCPOSLimit < 0, set bit in Status and disable CV ROC limiting.	Valid = 0.0 to maximum positive float Default = 0.0
CVROCNegLimit	REAL	CV decreasing rate of change limit, in percent per second. Rate of change limiting is only used when in Auto or CascadeRatio modes or Manual mode if CVManLimiting is TRUE. A value of zero disables CV ROC limiting. If value of CVROCNegLimit < 0, set bit in Status and disable CV ROC limiting.	Valid = 0.0 to maximum positive float Default = 0.0
HandFB	REAL	CV HandFeedback value. CV set to this value when in Hand mode and HandFBFault is FALSE (good health). This value would typically come from the output of a field mounted hand/auto station and would be used	Valid = 0.0...100.0 Default = 0.0

Input Parameters	Data Type	Description	Valid and Default Values
		to generate a bumpless transfer out of Hand mode. If value of HandFB < 0 or > 100, set unique Status bit and limit value used for CV.	
HandFBFault	BOOL	HandFB value bad health indicator. If the HandFB value is read from an analog input, then HandFBFault will normally be controlled by the status of the analog input channel. If HandFBFault is TRUE, it indicates an error on the input module, set bit in Status.	Default = FALSE FALSE = Good Health
WindupHIn	BOOL	Windup high request. When TRUE, CV will not be allowed to increase in value. This signal will typically be the WindupHOut output from a secondary loop.	Default = FALSE
WindupIIn	BOOL	Windup low request. When TRUE, CV will not be allowed to decrease in value. This signal will typically be the WindupLOut output from a secondary loop.	Default = FALSE
GainEUSpan	BOOL	ModelGain units in EU or as % of span. CV ModelGain units in EU or % of span. Set to interpret ModelGain as EU, reset to interpret ModelGain as % of Span.	Default = FALSE TRUE = Gain in EU FALSE = Gain in % of span
ProcessGainSign	BOOL	Used only for Autotuning. Sign of the process gain (Delta PV/Delta CV). Set to indicate a negative process gain (increase in output causes a decrease in PV).	Default = FALSE

Input Parameters	Data Type	Description	Valid and Default Values
		Reset to indicate a positive process gain (increase in output causes an increase in PV).	
ProcessType	DINT	Process type selection (1=Integrating, 0=non-integrating)	Default = 0
ModelGain	REAL	The internal model gain parameter. Enter a positive or negative gain depending on process direction.	Valid = maximum negative float –> maximum positive float Default = 0.0
ModelTC	REAL	The internal model time constant in seconds.	Valid = 0.0 to maximum positive float Default = 0.0
ModelDT	REAL	The internal model deadtime in seconds.	Valid = 0.0 to maximum positive float Default = 0.0
RespTC	REAL	The tuning parameter that determines the speed of the control variable action in seconds.	Valid = 0.0 to maximum positive float Default = 0.0
PVTracking	BOOL	SP track PV request. Set TRUE to enable SP to track PV. Ignored when in CascadeRatio or Auto modes.	Default = FALSE
CVTrackReq	BOOL	CV Track request. Set true to enable CV Tracking when autotune is OFF. Ignored in Hand and Override mode.	Default = FALSE
AllowCasRat	BOOL	Allow CascadeRatio mode permissive. Set TRUE to allow CascadeRatio mode to be selected by using either ProgCasRatReq or OperCasRatReq.	Default = FALSE
ManualAfterInit	BOOL	Manual mode after initialization request. When TRUE, the function block will be placed in the Manual mode when	Default = FALSE

Input Parameters	Data Type	Description	Valid and Default Values
		CVInitializing is set TRUE unless the current mode is Override or Hand. When ManualAfterInit is FALSE, the function block's mode will not be changed.	
ProgProgReq	BOOL	Program Program Request. Set TRUE by the user program to request Program control. Ignored if ProgOperReq is TRUE. Holding this TRUE and ProgOperReq FALSE can be used to lock the function block into program control. When ProgValueReset is TRUE, the function block resets the input to FALSE.	Default = FALSE
ProgOperReq	BOOL	Program Operator Request. Set TRUE by the user program to request Operator control. Holding this TRUE can be used to lock the function block into operator control. When ProgValueReset is TRUE, the function block resets the input to FALSE.	Default = FALSE
ProgCasRatReq	BOOL	Program-Cascade/Ratio mode request. Set TRUE by the user program to request Cascasde/Ratio mode. When ProgValueReset is TRUE, the function block resets the input to FALSE.	Default = FALSE
ProgAutoReq	BOOL	Program-Auto mode request. Set TRUE by the user program to request Auto mode. When ProgValueReset is TRUE, the function block resets the input to FALSE.	Default = FALSE
ProgManualReq	BOOL	Program-Manual mode request. Set TRUE by the	Default = FALSE

Input Parameters	Data Type	Description	Valid and Default Values
		user program to request Manual mode. When ProgValueReset is TRUE, the function block resets the input to FALSE.	
ProgOverrideReq	BOOL	Program-Override mode request. Set TRUE by the user program to request Override mode. When ProgValueReset is TRUE, the function block resets the input to FALSE.	Default = FALSE
ProgHandReq	BOOL	Program-Hand mode request. Set TRUE by the user program to request Hand mode. This value will usually be read as a digital input from a hand/auto station. When ProgValueReset is TRUE, the function block resets the input to FALSE.	Default = FALSE
OperProgReq	BOOL	Operator Program Request. Set TRUE by the operator interface to request Program control. The function block resets this parameter to FALSE.	Default = FALSE
OperOperReq		Operator Operator Request. Set TRUE by the operator interface to request Operator control. The function block will reset this parameter to FALSE.	Default = FALSE
OperCasRatReq	BOOL	Operator-CascadeRatio mode request. Set TRUE by the operator interface to request CascadeRatio mode. The function block will reset this parameter to FALSE.	Default = FALSE
OperAutoReq	BOOL	Operator-Auto mode request. Set TRUE by the operator interface to request Auto mode. The	Default = FALSE

Input Parameters	Data Type	Description	Valid and Default Values
		function block will reset this parameter to FALSE.	
OperManualReq	BOOL	Operator-Manual mode request. Set TRUE by the operator interface to request Manual mode. The function block will reset this parameter to FALSE.	Default = FALSE
ProgValueReset	BOOL	Reset Program control values. When TRUE, the Prog_xxx_Req inputs are reset to FALSE. When TRUE and Program control, set SPProg equal to SP and CVProg equal to CV. When ProgValueReset is TRUE, the function block resets this parameter to FALSE.	Default = FALSE
TimingMode	DINT	Selects Time Base Execution mode. Value/Description 0 = Periodic mode 1 = Oversample mode 2 = Real time sampling mode For more information about timing modes, see Function Block Attributes.	Valid = 0...2 Default = 0
OverSampleDT	REAL	Execution time for Oversample mode.	Valid = 0 to max. TON_Timer elapsed time (4194.303 seconds) Default = 0
RTSTime	DINT	Module update period for Real Time Sampling mode.	Valid = 1 to 32,767 1 count = 1 ms
RTSTimeStamp	DINT	Module time stamp value for Real Time Sampling mode.	Valid = 0 to 32,767 (wraps from 32,767 to 0) 1 count = 1 ms
PVTuneLimit	REAL	PV tuning limit scaled in the PV units. When	Range: any float

Input Parameters	Data Type	Description	Valid and Default Values
		Autotune is running and predicted PV exceeds this limit, the tuning will be aborted.	Default=0
AtuneTimeLimit	REAL	Maximum time for autotune to complete following the CV step change. When autotune exceeds this time, tuning will be aborted.	Valid range: any float > 0. Default = 60 minutes
NoiseLevel	DINT	An estimate of the noise level expected on the PV to compensate for it during tuning. The selections are: 0=low, 1=medium, 2=high	Range: 0 to 2 Default=1
CVStepSize	REAL	CV step size in percent for the tuning step test. Step size is directly added to CV subject to high/low limiting.	Range: -100% ... 100% Default=10%
ResponseSpeed	DINT	Desired speed of closed loop response. Slow response: ResponseSpeed=0 Medium response: ResponseSpeed=1 Fast response: ResponseSpeed=2. If ResponseSpeed is less than 0, Slow response is used. If ResponseSpeed is greater than 2, Fast response is used.	Range: 0 to 2 Default=1
ModelInit	BOOL	Internal model initialization switch.	Default = FALSE
Factor	REAL	Non-integrating model approximation factor. Only used for integrating process types.	Default = 100
AtuneStart	BOOL	Start Autotune request. Set True to initiate auto tuning of the function	Default = FALSE

Input Parameters	Data Type	Description	Valid and Default Values
		block. Ignored when IMC is not in Manual mode. The function block will reset this parameter to FALSE.	
AtuneUseModel	BOOL	Use Autotune model request. Set True to replace the current model parameters with the calculated Autotune model parameters. The function block sets the input parameter to FALSE.	Default = FALSE
AtuneAbort	BOOL	Abort Autotune request. Set True to abort the auto tuning of the IMC function block. The function block sets input parameter to FALSE.	Default = FALSE

Output Parameters	Data Type	Description	Valid and Default Values
EnableOut	BOOL	Indicates if instruction is enabled. Cleared to false if CVEU overflows.	
CVEU	REAL	Scaled control variable output. Scaled by using CVEUMax and CVEUMin, where CVEUMax corresponds to 100% and CVEUMin corresponds to 0%. This output is typically used to control an analog output module or a secondary loop. CVEU = (CV * CVEUSpan / 100) + CVEUMin CVEU span calculation: CVEUSpan = (CVEUMax – CVEUMin)	
CV	REAL	Control variable output. This value will always be expressed as 0...100%. CV is limited by CVHLimit and CVLLimit when in Auto or CascadeRatio	

Output Parameters	Data Type	Description	Valid and Default Values
		mode or Manual mode if CVManLimiting is TRUE; otherwise limited by 0 and 100%.	
DeltaCV	REAL	Difference between the Current CV and the previous CV (Current CV - previous CV).	
CVInitializing	BOOL	Initialization mode indicator. Set TRUE when CVInitReq or function block FirstScan are TRUE, or on a TRUE to FALSE transition of CVFault (bad to good). CVInitializing is set FALSE after the function block has been initialized and CVInitReq is no longer TRUE.	
CVHAlarm	BOOL	CV high alarm indicator. TRUE when the calculated value for CV > 100 or CVHLimit.	
CVLAlarm	BOOL	CV low alarm indicator. TRUE when the calculated value for CV < 0 or CVLLimit.	
CVROCPosAlarm	BOOL	CV rate of change alarm indicator. TRUE when the calculated rate of change for CV exceeds CVROCPosLimit.	
CVROCNegAlarm	REAL	CV rate of change alarm indicator. TRUE when the calculated rate of change for CV exceeds CVROCNegLimit.	
SP	REAL	Current setpoint value. The value of SP is used to control CV when in the Auto, the CascadeRatio, or the PV Tracking mode, scaled in PV units.	

Output Parameters	Data Type	Description	Valid and Default Values
SPPercent	REAL	The value of SP expressed in percent of span of PV. SPPercent = ((SP – PVEUMin) * 100) / PVSpan where PVSpan = PVEUMax – PVEUMin	
SPHAlarm	BOOL	SP high alarm indicator. TRUE when the SP ≥ SPHLimit.	
SPLAlarm	BOOL	SP low alarm indicator. TRUE when the SP ≤ SPLLimit.	
PVPercent	REAL	PV expressed in percent of span. PVPercent = ((PV – PVEUMin) * 100) / PVSpan PV Span calculation: PVSpan = (PVEUMax – PVEUMin)	
E	REAL	Process error. Difference between SP and PV, scaled in PV units.	
EPercent	REAL	The error expressed as a percent of span.	
InitPrimary	BOOL	Initialize primary loop command. TRUE when not in CasRat mode or when CVInitializing is TRUE. This signal normally would be used by the CVInitReq input of a primary loop.	
WindupHOut	BOOL	Windup high indicator. TRUE when either a SP high or CV high/low limit has been reached. This signal will typically be used by the WindupHIn input to limit the windup of the CV output on a primary loop.	
WindupLOut	BOOL	Windup low indicator. TRUE when either a SP	

Output Parameters	Data Type	Description	Valid and Default Values
		or CV high/low limit has been reached. This signal will typically be used by the WindupLIn input to limit the windup of the CV output on a primary loop.	
Ratio	REAL	Current ratio multiplier, no units.	
RatioHAlarm	BOOL	Ratio high alarm indicator. TRUE when Ratio > RatioHLimit.	
RatioLAlarm	BOOL	Ratio low alarm indicator. TRUE when Ratio < RatioLLimit.	
ProgOper	BOOL	Program/Operator control indicator. TRUE when in Program control. FALSE when in Operator control.	
CasRat	BOOL	CascadeRatio mode indicator. TRUE when in the CascadeRatio mode.	
Auto	BOOL	Auto mode indicator. TRUE when in the Auto mode.	
Manual	BOOL	Manual mode indicator. TRUE when in the Manual mode.	
Override	BOOL	Override mode indicator. TRUE when in the Override mode.	
Hand	BOOL	Hand mode indicator. TRUE when in the Hand mode.	
DeltaT	REAL	Elapsed time between updates in seconds.	
StepSizeUsed	REAL	Actual CV step size used for tuning.	
GainTuned	REAL	The calculated value of the internal model gain after tuning is completed.	

Output Parameters	Data Type	Description	Valid and Default Values
TCTuned	REAL	The calculated value of the internal model time constant after tuning is completed.	
DTTuned	REAL	The calculated value of the internal model deadtime after tuning is completed.	
RespTCTunedS	REAL	The calculated value of the control variable time constant in slow response speed after tuning is completed.	
RespTCTunedM	REAL	The calculated value of the control variable time constant in medium response speed after tuning is completed.	
RespTCTunedF	REAL	The calculated value of the control variable time constant in fast response speed after tuning is completed.	
AtuneOn	BOOL	Set True when auto tuning has been initiated.	
AtuneDone	BOOL	Set True when auto tuning has completed successfully.	
AtuneAborted	BOOL	Set True when auto tuning has been aborted by user or due to errors that occurred during the auto tuning operation.	
AtuneStatus	DINT	Indicates the block tuning status.	
AtuneFault	BOOL	Autotune has generated any of the following faults.	Bit 0 of AtuneStatus
AtunePVOutOfLimit	BOOL	Either PV or the deadtime-step ahead prediction of PV exceeds PVTuneLimit during	Bit 1 of AtuneStatus

Output Parameters	Data Type	Description	Valid and Default Values
		Autotuning. When True, Autotuning is aborted.	
AtuneModeInv	BOOL	The IMC mode was not Manual at start of Autotuning or the IMC mode was changed from Manual during Autotuning. When True, Autotuning is not started or is aborted.	Bit 2 of AtuneStatus
AtuneCVWindupFault	BOOL	WindupHIn or WindupLIn is True at start of Autotuning or during Autotuning. When True, Autotuning is not started or is aborted.	Bit 3 of AtuneStatus
AtuneStepSize0	BOOL	StepSizeUsed = 0 at start of Autotuning. When True, Autotuning is not started.	Bit 4 of AtuneStatus
AtuneCVLimitsFault	BOOL	CVLimitsInv and CVManLimiting are True at start of Autotuning or during Autotuning. When True, Autotuning is not started or is aborted.	Bit 5 of AtuneStatus
AtuneCVInitFault	BOOL	CVInitializing is True at start of Autotuning or during Autotuning. When True, Autotuning is not started or is aborted.	Bit 6 of AtuneStatus
AtuneEUSpanChanged	BOOL	CVEUSpan or PVEUSpan changes during Autotuning. When True, Autotuning is aborted.	Bit 7 of AtuneStatus
AtuneCVChanged	BOOL	CVOper is changed when in Operatior control or CVProg is changed when in Program control or CV becomes high/low or ROC limited during Autotuning. When True, Autotuning is aborted.	Bit 8 of AtuneStatus
AtuneTimeout	BOOL	Elapsed time is greater then AtuneTimeLimit since step test is started.	Bit 9 of AtuneStatus

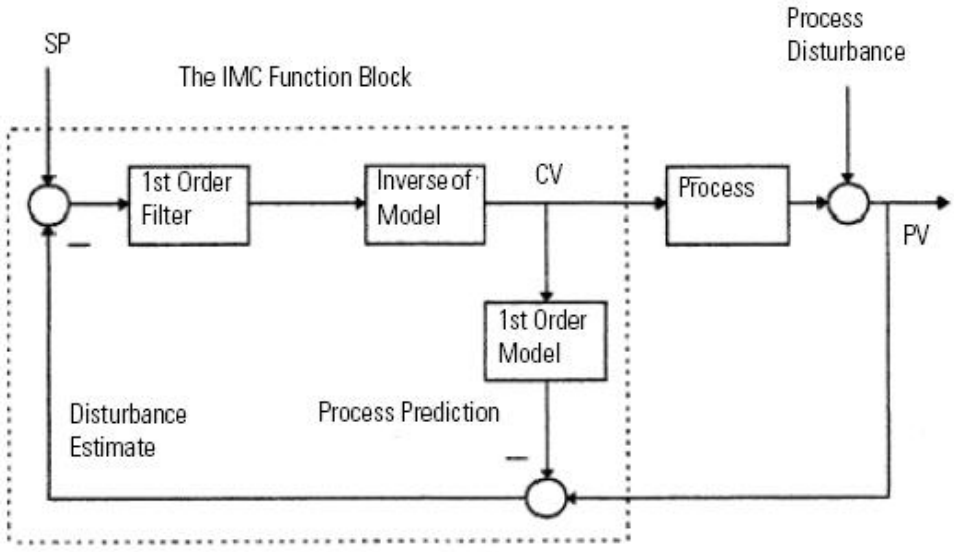
Output Parameters	Data Type	Description	Valid and Default Values
		When True, Autotuning is aborted.	
AtunePVNotSettled	BOOL	The PV is changed too much to Autotune. When True, Autotuning is aborted. Wait until PV is more stable before autotuning.	Bit 10 of AtuneStatus
Status1	DINT	Bit mapped status of the function block.	
Status2	DINT	Additional bit mapped status for the function block.	
InstructFault	BOOL	Function block has generated a fault. Indicates state of bits in Status1 and status2. A value of 0 indicates that no faults have occurred. Any parameters that could be configured with an invalid value must have a status parameter to indicate their invalid status.	Bit 0 of Status1
PVFaulted	BOOL	Process variable PV health bad.	Bit 1 of Status1
CVFaulted	BOOL	Control variable CV faulted	Bit 2 of Status1
HandFBFaulted	BOOL	HandFB value health bad	Bit 3 of Status1
PVSpanInv	BOOL	The span of PV invalid, PVEUMax < PVEUMin.	Bit 4 of Status1
SPProgInv	BOOL	SPProg < SPLLimit or > SPHLimit. Limit value used for SP.	Bit 5 of Status1
SPOperInv	BOOL	SPOper < SPLLimit or > SPHLimit. Limit value used for SP.	Bit 6 of Status1
SPCascadeInv	BOOL	SPCascade < SPLLimit or > SPHLimit. Limit value used for SP.	Bit 7 of Status1

Output Parameters	Data Type	Description	Valid and Default Values
SPLimitsInv	BOOL	Limits invalid: SPLLimit < PVEUMin, SPHLimit > PVEUMax, or SPHLimit < SPLLimit. If SPHLimit < SPLLimit, then limit value using SPLLimit.	Bit 8 of Status1
RatioLimitsInv	BOOL	Ratio high-low limits invalid, low limit < 0 or high limit < low limit.	Bit 9 of Status1
RatioProgInv	BOOL	RatioProg < RatioLLimit or > RatioHLimit. Limit value used for Ratio.	Bit 10 of Status1
RatioOperInv	BOOL	RatioOper < RatioLLimit or > RatioHLimit. Limit value used for Ratio.	Bit 11 of Status1
CVProgInv	BOOL	CVProg < 0 or > 100, or < CVLLimit or > CVHLimit when CVManLimiting is TRUE. Limit value used for CV.	Bit 12 of Status1
CVOperInv	BOOL	CVOper < 0 or > 100, or < CVLLimit or > CVHLimit when CVManLimiting is TRUE. Limit value used for CV.	Bit 13 of Status1
CVOverrideValueInv	BOOL	CVOverrideValue < 0 or > 100. Limit value used for CV.	Bit 14 of Status1
CVTrackValueInv	BOOL	CVTrackValue < 0 or > 100. Limit value used for CV.	Bit 15 of Status1
CVEUSpanInv	BOOL	The span of CVEU invalid, CVEUMax equals CVEUMin.	Bit 16 of Status1
CVLimitsInv	BOOL	CVLLimit < 0, CVHLimit > 100, or CVHLimit <= CVLLimit. If CVHLimit <= CVLLimit, limit CV by using CVLLimit.	Bit 17 of Status1
CVROCLimitInv	BOOL	CVROCLimit < 0, disables ROC limiting.	Bit 18 of Status1

Output Parameters	Data Type	Description	Valid and Default Values
HandFBInv	BOOL	HandFB < 0 or > 100. Limit value used for CV.	Bit 19 of Status1
SampleTimeTooSmall	BOOL	Model DeadTime / DeltaT must be less than or equal to 200.	Bit 20 of Status1
FactorInv	BOOL	Factor < 0.	Bit 21 of Status1
ModuleGainInv	BOOL	ModelGain is 1.#QNAN or -1.#IND (Not A Number), or ±1.$ (Infinity ∞)	Bit 22 of Status1
ModelTCInv	BOOL	ModelTC < 0.	Bit 23 of Status1
ModelDTInv	BOOL	ModelDT < 0.	Bit 24 of Status1
RespTCInv	BOOL	RespTC < 0.	Bit 25 of Status1
TimingModelInv	BOOL	TimingMode invalid. If the current mode is not Override or Hand then set to Manual mode.	Bit 27 of Status2
RTSMissed	BOOL	Only used when in Real Time Sampling mode. Is TRUE when ABS(DeltaT - RTSTime) > 1 millisecond.	Bit 28 of Status2.
RTSTimeInv	BOOL	RTSTime invalid.	Bit 29 of Status2.
RTSTimeStampInv	BOOL	RTSTimeStamp invalid. If the current mode is not Override or Hand then set to Manual mode.	Bit 30 of Status2.
DeltaTInv	BOOL	DeltaT invalid. If the current mode is not Override or Hand then set to Manual mode.	Bit 31 of Status2.

Description

The following illustration shows the IMC function block configuration.



At each execution, the IMC function block compares the actual PV measurement with PV prediction. The result is called disturbance estimate, which is the effect of unmeasured process disturbances combined with the modeling imprecision. The disturbance estimate is used as a bias for the setpoint of the control variable. In the ideal case of no disturbances and perfect modeling, the disturbance estimate (the feedback signal) becomes equal to zero.

First Order Model	M = K/(T*s+1)*exp(-D*s)	
Inverse of Model		Inv = (T*s+1)/K
First Order Filter		F = 1/(e*s+1)

PV prediction = exp(-D*s)/(e*s+1) * (SP - Dist. estimate)

K...	Model gain
T...	Model time constant
D...	Model deadtime
e...	Response time constant
s...	Laplace variable

The function block then calculates the CV value (CVHLimit, CVLLimit, and rate of change limits are imposed) and the PV prediction.

The IMC function block can be used in place of a PID function block with the advantage over the PID control variable when controlling processes with large deadtimes.

For an integrating process type (such as level control and position control), an internal nonintegrating model is used to approximate the integrating process. The Factor parameter is

used to convert the identified integrating-process model to a nonintegrating internal model that is used for CV calculation. This is necessary to provide for stable IMC execution.

Affects Math Status Flags

No

Major/Minor Faults

None specific to this instruction. See Common Attributes on page for operand-related faults.

Execution

Function Block

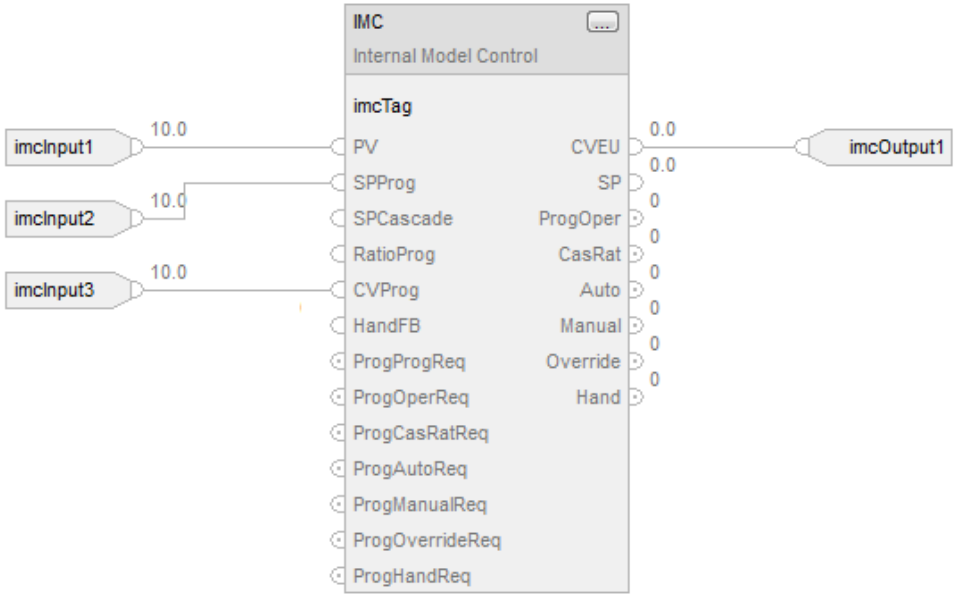
Condition/State	Action Taken
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is true	EnableIn and .EnableOut bits are set to true. The instruction executes.
Instruction first run	N/A
Instruction first scan	N/A
Postscan	EnableIn and EnableOut bits are cleared to false.

Structured Text

Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.
Normal Execution	See Tag.EnableIn is true in the Function Block table.
Postscan	See Postscan in the Function Block table.

Examples

Function Block



Structured Text

```
imcTag.PV := imcInput1;
imcTag.SPProg := imcInput2;
imcTag.CVProg := imcInput3;
IMC(imcTag);
imcOutput1 := imcTag.CVEU;
```

IMC Function Block Configuration

Follow these steps to create a basic IMC configuration.

1. Starting with the default configuration, configure the following parameters.

Parameter	Description
PVEUMax	Maximum scaled value for PV.
PVEUMin	Minimum scaled value for PV.
SPHLimit	SP high limit value, scaled in PV units.
SPLLimit	SP low limit value, scaled in PV units.
CVInitValue	An initial value of the control variable output.

2. If you have the process model available, you can intuitively tune the IMC control variable by entering the following four parameters.

Parameter	Description
Model Gain	A nonzero number (negative for direct acting control variable. positive for reverse acting control variable).
Model Time Constant	Always a positive number.
Model Deadtime	Always a positive number.
Response Time Constant	Always a positive number - used to tune the response of the IMC control variable. A smaller number gives a faster response.

At this point, you have completed the basic configuration. You did not configure the built-in tuner. The control variable is ready to be put online in either Auto or Manual mode. For tuning, use the default settings. Refer to IMC Function Block Tuning on page 235.

3. If you do not know the process model, you need to identify the model and tune the control variable by using the built-in tuner (modeler) for the control variable to operate correctly in the Auto mode.

   The control variable uses a first order lag with deadtime internal process model and a first order filter (total of four tuning parameters) to calculate the CV. The CV is calculated such that the process variable (PV) follows a first order lag trajectory when approaching the setpoint value.

   Speed of response depends on the value of the response time constant. The smaller that the response time constant is, the faster the control variable response will be. The response time constant should be set such that the PV reaches the setpoint in a reasonable time based on the process dynamics. The larger that the response time constant is, the slower the control variable response will be, but the control variable also becomes more robust. Refer to IMC Function Block Tuning.

   In the Manual mode, the CV is set equal to the operator-entered or program-generated CVOper or CVProg parameter.

   For the Manual to Auto mode bumpless transfer and for safe operation of the control variable, the CV rate of change limiter is implemented such that the CV cannot change from its current state any faster than the rate of change limit parameter specified.

4. Set the CVROCPosLimit and CVROCNegLimit to limit the CV rate of change.

   Rate limiting is not imposed when the control variable is in Manual mode unless CVManLimiting is set.

IMC Function Block Initialize

A Model Initialization occurs:

- during First Scan of the block
- when the ModelInit request parameter is set
- when DeltaT changes

You may need to manually adjust the internal model parameters or the response time constants. You can do so by changing the appropriate parameters and setting the appropriate ModelInit bit. The internal states of the function block will be initialized, and the bit will automatically reset.

For example, if you modify the IMC function block Model Gain for CV - PV, set the ModelInit parameter to TRUE to initialize the CV - PV internal model parameters and for the new model gain to take effect.

IMC Function Block Tuning

The function block is equipped with an internal tuner (modeler). The purpose of the tuner is to identify the process model parameters and to use these parameters as internal model parameters (gain, time constant, and deadtime). The tuner also calculates an optimal response-time constant.

Set the tuner by configuring the following parameters.

ProcessType	Integral (level, position control) or nonintegrating (flow, pressure control)
ProcessGainSign	Set to indicate a negative process gain (increase in output causes a decrease in PV); reset to indicate a positive process gain (increase in output causes an increase in PV).
ResponseSpeed	Slow, medium, or fast, based on control variable.
NoiseLevel	An estimate of noise level on PV-low, medium, or high such that the tuner can distinguish which PV change is a random noise and which is caused by the CV step change.
StepSize	A nonzero positive or negative number defining the magnitude of CV step change in either positive or negative direction, respectively.
PVTuneLimit	(Only for integrating process type) in PV engineering units, defines how much of PV change that is caused by CV change to tolerate before aborting the tuning test due to exceeding this limit.

The tuner is started by setting the AtuneStart bit. You can stop the tuning by setting the AtuneAbort bit. After the tuning is completed successfully, the GainTuned, TCTuned, DTTuned, and RespTCTuned parameters are updated with the tuning results, and the AtuneStatus code is set to indicate complete.

You can copy these parameters to the ModelGain, ModelTC, and ResponseTC, respectively, by setting the AtuneUseModel bit. The function block will automatically initialize the internal variables and continue normal operation. It will automatically reset the AtuneUseModel bit.

IMC Function Block Tuning Errors

If an error occurs during the tuning procedure, the tuning is aborted, and the AtuneStatus bit is set. You can abort the tuning by setting the AtuneAbort bit.

After an abort, the CV will assume its value before the step change, and the GainTuned, TCTuned, DTTuned, and RespTCTuned parameters are not updated. The AtuneStatus parameter identifies the reason for the abort.

IMC Function Block Tuning Procedure

Follow these steps to configure the tuner.

1. Put the CV into Manual mode.
2. Set the AtuneStart parameter.

   The tuner starts collecting PV and CV data for noise calculation.
3. After collecting 60 samples (60*DeltaT) period, the tuner adds StepSize to the CV.

   After successfully collecting the PV data as a result of the CV step change, the CV assumes its value before the step change and the AtuneStatus, GainTuned, TCTuned, DTTuned, and RespTCTuned parameters are updated.
4. Set the AtuneUseModel parameter to copy the tuned parameters to the model parameters.

   The function block then resets the AtuneUseModel parameter.

   After a successful AutoTuneDone, the Atune parameter is set to one (1). Tuning completed successfully.

Modular Multivariable Control (MMC)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, Compact GuardLogix 5380, CompactLogix 5380, ControlLogix 5580, GuardLogix 5580, and ControlLogix 5590 controllers.

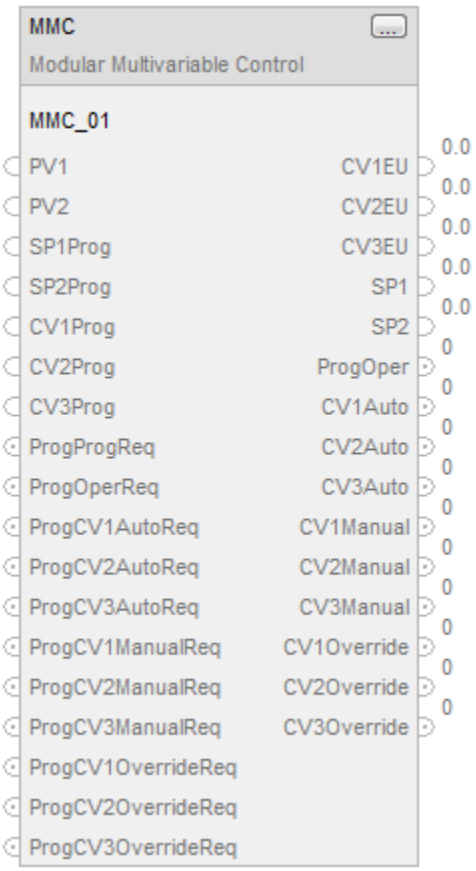
The Modular Multivariable Control (MMC) instruction controls two process variables to their setpoints using up to three control variables. The MMC instruction calculates the control variables (CV1, CV2, and CV3) in the auto mode based on the PV1 -SP1, PV2 - SP2 deviation, internal model, and tuning.

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram logic.

Function Block



Structured Text

MMC(MMC_tag);

Operands

Function Block

Operands:	Type	Format	Description
MMC tag	MODULAR MULTIVARIABLE CONTROL	structure	MMC Structure

Structured Text

Operands:	Type	Format	Description
MMC tag	MODULAR MULTIVARIABLE CONTROL	structure	MMC Structure

Structure

The following table describes the input parameters in the MMC function block.

Input Parameters	Data Type	Description	Values
EnableIn	BOOL	Enable Input. If False, the function block will not execute and outputs are not updated.	Default=TRUE
PV1	REAL	Scaled process variable input 1. This value is typically read from an analog input module.	Valid = any float Default = 0.0
PV2	REAL	Scaled process variable input 2. This value is typically read from an analog input module.	Valid = any float Default = 0.0
PV1Fault	BOOL	PV1 bad health indicator. If PV1 is read from an analog input, then PV1Fault will normally be controlled by the analog input fault status. If PVFault is TRUE, it indicates an error on the input module, set bit in Status.FALSE = Good Health	Default = FALSE
PV2Fault	BOOL	PV2 bad health indicator. If PV2 is read from an analog input, then PV2Fault will normally be controlled by the analog input fault status. If PVFault is TRUE, it indicates an error on the input module, set bit in Status.FALSE = Good Health	Default = FALSE
PV1EUMax	REAL	Maximum scaled value for PV1. The value of PV1 and SP1 that corresponds to 100% span of the Process Variable. If PV1EUMax ≤ PV1EUMin, set bit in Status.	Valid = PV1EUMin < PV1EUMax <= maximum positive float Default = 100.0
PV2EUMax	REAL	Maximum scaled value for PV2. The value of PV2 and SP2 that corresponds to 100% span of the Process Variable. If PV2EUMax ≤ PV2EUMin, set bit in Status.	Valid = PV2EUMin < PV2EUMax <= maximum positive float Default = 100.0

Input Parameters	Data Type	Description	Values
PV1UEMin	REAL	Minimum scaled value for PV1. The value of PV1 and SP1 that corresponds to 0% span of the Process Variable. If PV1EUMax ≤ PV1EUMin, set bit in Status.	Valid = maximum negative float <= PV1EUMin < PV1EUMax Default = 0.0
PV2UEMin	REAL	Minimum scaled value for PV2. The value of PV2 and SP2 that corresponds to 0% span of the Process Variable. If PV1EUMax ≤ PV1EUMin, set bit in Status.	Valid = maximum negative float <= PV2EUMin < PV2EUMax Default = 0.0
SP1Prog	REAL	SP1 Program value, scaled in PV units. SP1 is set to this value when Program control.	Valid = SP1LLimit to SP1HLimit Default = 0.0
SP2Prog	REAL	SP2 Program value, scaled in PV units. SP2 is set to this value when Program control.	Valid = SP2LLimit to SP2HLimit Default = 0.0
SP1Oper	REAL	SP1 Operator value, scaled in PV units. SP1 set to this value when Operator control. If value of SP1Prog or SP1Oper < SP1LLimit or > SP1HLimit, set bit in Status and limit value used for SP.	Valid = SP1LLimit to SP1HLimit Default = 0.0
SP2Oper	REAL	SP2 Operator value, scaled in PV units. SP2 set to this value when Operator control. If value of SP2Prog or SP2Oper < SP2LLimit or > SP2HLimit, set bit in Status and limit value used for SP.	Valid = SP2LLimit to SP2HLimit Default = 0.0
SP1HLimit	REAL	SP1 high limit value, scaled in PV units. • If SP1LLimit < PV1EUMin, or SP1HLimit > PV1EUMax, set bit in Status.	Valid = SP1LLimit to PV1EUMax Default = 100.0

Input Parameters	Data Type	Description	Values
		• If SP1HLimit < SP1LLimit, set bit in Status and limit SP by using the value of S1PLLimit.	
SP2HLimit	REAL	SP2 high limit value, scaled in PV units. • If SP2LLimit < PV2EUMin, or SP2HLimit > PV2EUMax, set bit in Status. • If SP2HLimit < SP2LLimit, set bit in Status and limit SP by using the value of SPLLimit.	Valid = SP2LLimit to PV2EUMax Default = 100.0
SP1LLimit	REAL	SP1 low limit value, scaled in PV units. • If SP1LLimit < PV1EUMin, or SP1HLimit > PV1EUMax, set bit in Status. • If SP1HLimit < SP1LLimit, set bit in Status and limit SP by using the value of SPLLimit.	Valid = PV1EUMin to SP1HLimit Default = 0.0
SP2LLimit	REAL	SP2 low limit value, scaled in PV units. • If SP2LLimit < PV2EUMin, or SP2HLimit > PV2EUMax, set bit in Status. • If SP2HLimit < SP2LLimit, set bit in Status and limit SP by using the value of SPLLimit.	Valid = PV2EUMin to SP2HLimit Default = 0.0
CV1Fault	BOOL	Control variable 1 bad health indicator. If CV1EU controls an analog output, then CV1Fault will normally	Default = FALSE

Input Parameters	Data Type	Description	Values
		come from the analog output's fault status. If CV1Fault is TRUE, it indicates an error on the output module, set bit in Status.FALSE = Good Health	
CV2Fault	BOOL	Control variable 2 bad health indicator. If CV2EU controls an analog output, then CV2Fault will normally come from the analog output's fault status. If CV2Fault is TRUE, it indicates an error on the output module, set bit in Status.FALSE = Good Health	Default = FALSE
CV3Fault	BOOL	Control variable 3 bad health indicator. If CV3EU controls an analog output, then CV3Fault will normally come from the analog output's fault status. If CV3Fault is TRUE, it indicates an error on the output module, set bit in Status.FALSE = Good Health	Default = FALSE
CV1InitReq	BOOL	CV1 initialization request. While TRUE, set CV1EU to the value of CV1InitValue. This signal will normally be controlled by the In Hold status on the analog output module controlled by CV1EU or from the InitPrimary output of a secondary loop. The instruction initialization is disabled when CV1Faulted or CV1EUSpanInv are TRUE.	Default = FALSE
CV2InitReq	BOOL	CV2 initialization request. While TRUE, set CV2EU to the value of CV2InitValue.	Default = FALSE

Input Parameters	Data Type	Description	Values
		This signal will normally be controlled by the In Hold status on the analog output module controlled by CV2EU or from the InitPrimary output of a secondary loop. The instruction initialization is disabled when CV2Faulted or CV2EUSpanInv are TRUE.	
CV3InitReq	BOOL	CV3 initialization request. While TRUE, set CV3EU to the value of CV3InitValue. This signal will normally be controlled by the In Hold status on the analog output module controlled by CV3EU or from the InitPrimary output of a secondary loop. The instruction initialization is disabled when CV3Faulted or CV3EUSpanInv are TRUE.	Default = FALSE
CV1InitValue	REAL	CV1EU initialization value, scaled in CV1EU units. When CV1Initializing is TRUE set CV1EU equal to CV1InitValue and CV1 to the corresponding percentage value. CV1InitValue will normally come from the feedback of the analog output controlled by CV1EU or from the setpoint of a secondary loop. The instruction initialization is disabled when CV1Faulted or CV1EUSpanInv are TRUE.	Valid = any float Default = 0.0
CV2InitValue	REAL	CV2EU initialization value, scaled in CV2EU units. When CV2Initializing is TRUE set CV2EU equal to CV2InitValue and CV2 to the corresponding	Valid = any float Default = 0.0

Input Parameters	Data Type	Description	Values
		percentage value. CV2InitValue will normally come from the feedback of the analog output controlled by CV2EU or from the setpoint of a secondary loop. The instruction initialization is disabled when CV2Faulted or CV2EUSpanInv are TRUE.	
CV3InitValue	REAL	CV3EU initialization value, scaled in CV3EU units. When CV3Initializing is TRUE set CV3EU equal to CV3InitValue and CV3 to the corresponding percentage value. CV3InitValue will normally come from the feedback of the analog output controlled by CV3EU or from the setpoint of a secondary loop. The instruction initialization is disabled when CV3Faulted or CV3EUSpanInv are TRUE.	Valid = any float Default = 0.0
CV1Prog	REAL	CV1 Program-Manual value. CV1 is set to this value when in Program control and Manual mode.	Valid = 0.0 through 100.0 Default = 0.0
CV2Prog	REAL	CV2 Program-Manual value. CV2 is set to this value when in Program control and Manual mode.	Valid = 0.0 through 100.0 Default = 0.0
CV3Prog	REAL	CV3 Program-Manual value. CV3 is set to this value when in Program control and Manual mode.	Valid = 0.0 through 100.0 Default = 0.0
CV1Oper	REAL	CV1 Operator-Manual value. • CV1 is set to this value when in Operator control and Manual mode. If not	Valid = 0.0 through 100.0 Default = 0.0

Input Parameters	Data Type	Description	Values
		Operator-Manual mode, set CV1Oper to the value of CV1 at the end of each function block execution. • If value of CV1Prog or CV1Oper < 0 or > 100, or < CV1LLimit or > CV1HLimit when CVManLimiting is TRUE, set unique Status bit and limit value used for CV1.	
CV2Oper	REAL	CV2 Operator-Manual value. • CV2 is set to this value when in Operator control and Manual mode. If not Operator-Manual mode, set CV2Oper to the value of CV2 at the end of each function block execution. • If value of CV2Prog or CV2Oper < 0 or > 100, or < CV2LLimit or > CV2HLimit when CVManLimiting is TRUE, set unique Status bit and limit value used for CV2.	Valid = 0.0 through 100.0 Default = 0.0
CV3Oper	REAL	CV3 Operator-Manual value. • CV3 is set to this value when in Operator control and Manual mode. If not Operator-Manual mode, set CV3Oper to the value of CV3 at the end of each function block execution. • If value of CV3Prog or CV3Oper < 0 or > 100, or < CV3LLimit or > CV3HLimit when	Valid = 0.0 through 100.0 Default = 0.0

Input Parameters	Data Type	Description	Values
		CVManLimiting is TRUE, set unique Status bit and limit value used for CV3.	
CV1OverrideValue	REAL	CV1 Override value. • CV1 set to this value when in Override mode. This value should correspond to a safe state output of the loop. • If value of CV1OverrideValue < 0 or >100, set unique Status bit and limit value used for CV1.	Valid = 0.0 through 100.0 Default = 0.0
CV2OverrideValue	REAL	CV2 Override value. • CV2 set to this value when in Override mode. This value should correspond to a safe state output of the loop. • If value of CV2OverrideValue < 0 or >100, set unique Status bit and limit value used for CV2.	Valid = 0.0 through 100.0 Default = 0.0
CV3OverrideValue	REAL	CV3 Override value. CV3 set to this value when in Override mode. This value should correspond to a safe state output of the loop. If value of CV3OverrideValue < 0 or >100, set unique Status bit and limit value used for CV3.	Valid = 0.0 through 100.0 Default = 0.0
CVManLimiting	BOOL	Limit CV(n), where (n) can be 1, 2, or 3, in Manual mode. If Manual mode and CVManLimiting is TRUE, CV(n) will be limited	Default = FALSE

Input Parameters	Data Type	Description	Values
		by the CV(n)HLimit and CV(n)LLimit values.	
CV1EUMax	REAL	Maximum value for CV1EU. The value of CV1EU that corresponds to 100% CV1. If CVEUMax = CVEUMin, set bit in Status.	Valid = any float Default = 100.0
CV2EUMax	REAL	Maximum value for CV2EU. The value of CV2EU that corresponds to 100% CV2. If CVEUMax = CVEUMin, set bit in Status.	Valid = any float Default = 100.0
CV3EUMax	REAL	Maximum value for CV3EU. The value of CV3EU that corresponds to 100% CV3. If CVEUMax = CVEUMin, set bit in Status.	Valid = any float Default = 100.0
CV1EUMin	REAL	Minimum value of CV1EU. The value of CV1EU that corresponds to 0% CV1. If CVEUMax = CVEUMin, set bit in Status.	Valid = any float Default = 0.0
CV2EUMin	REAL	Minimum value of CV2EU. The value of CV2EU that corresponds to 0% CV2. If CVEUMax = CVEUMin, set bit in Status.	Valid = any float Default = 0.0
CV3EUMin	REAL	Minimum value of CV3EU. The value of CV3EU that corresponds to 0% CV3. If CVEUMax = CVEUMin, set bit in Status.	Valid = any float Default = 0.0
CV1HLimit	REAL	CV1 high limit value. This is used to set the CV1HAlarm output. It is also used for limiting CV1 when in Auto mode or in Manual if CVManLimiting is TRUE. • If CV1LLimit < 0, if CV1HLimit > 100, if CV1HLimit < CV1LLimit, set bit in Status.	Valid = CV1LLimit < CV1HLimit ≤ 100.0 Default = 100.0

Input Parameters	Data Type	Description	Values
		• If CV1HLimit < CV1LLimit, limit CV1 by using the value of CV1LLimit.	
CV2HLimit	REAL	CV2 high limit value. This is used to set the CV2HAlarm output. It is also used for limiting CV2 when in Auto mode or in Manual if CVManLimiting is TRUE. • If CV2LLimit < 0, if CV2HLimit > 100, if CV2HLimit < CV2LLimit, set bit in Status. • If CV2HLimit < CV2LLimit, limit CV2 by using the value of CV2LLimit.	Valid = CV2LLimit < CV2HLimit ≤ 100.0 Default = 100.0
CV3HLimit	REAL	CV3 high limit value. This is used to set the CV3HAlarm output. It is also used for limiting CV3 when in Auto mode or in Manual if CVManLimiting is TRUE. • If CV3LLimit < 0, if CV3HLimit > 100, if CV3HLimit < CV3LLimit, set bit in Status. • If CV3HLimit < CV3LLimit, limit CV3 by using the value of CV3LLimit.	Valid = CV3LLimit < CV3HLimit ≤ 100.0 Default = 100.0
CV1LLimit	REAL	CV1 low limit value. This is used to set the CV1LAlarm output. It is also used for limiting CV1 when in Auto mode or in Manual mode if CVManLimiting is TRUE. • If CV1LLimit < 0, if CV1HLimit > 100, if CV1HLimit < CV1LLimit, set bit in Status.	Valid = 0.0 ≤ CV1LLimit < CV1HLimit Default = 0.0

Input Parameters	Data Type	Description	Values
		• If CV1HLimit < CV1LLimit, limit CV1 by using the value of CV1LLimit.	
CV2LLimit	REAL	CV2 low limit value. This is used to set the CV2LAlarm output. It is also used for limiting CV2 when in Auto mode or in Manual mode if CVManLimiting is TRUE. • If CV2LLimit < 0, if CV2HLimit > 100, if CV2HLimit < CV2LLimit, set bit in Status. • If CV2HLimit < CV2LLimit, limit CV2 by using the value of CV2LLimit.	Valid = 0.0 ≤ CV2LLimit < CV1HLimit Default = 0.0
CV3LLimit	REAL	CV3 low limit value. This is used to set the CV3LAlarm output. It is also used for limiting CV3 when in Auto mode or in Manual mode if CVManLimiting is TRUE. • If CV3LLimit < 0, if CV3HLimit > 100, if CV3HLimit < CV3LLimit, set bit in Status. • If CV3HLimit < CV3LLimit, limit CV by using the value of CV3LLimit.	Valid = 0.0 ≤ CV3LLimit < CV1HLimit Default = 0.0
CV1ROCPosLimit	REAL	CV1 rate of change limit, in percent per second. Rate of change limiting is only used when in Auto mode or in Manual mode if CVManLimiting is TRUE. A value of zero disables CV1 ROC limiting. If value of CV1ROCLimit < 0, set bit in Status and disable CV1 ROC limiting.	Valid = 0.0 to maximum positive float Default = 0.0

Input Parameters	Data Type	Description	Values
CV2ROCPosLimit	REAL	CV2 rate of change limit, in percent per second. Rate of change limiting is only used when in Auto mode or in Manual mode if CVManLimiting is TRUE. A value of zero disables CV2 ROC limiting. If value of CV2ROCLimit < 0, set bit in Status and disable CV2 ROC limiting.	Valid = 0.0 to maximum positive float Default = 0.0
CV3ROCPosLimit	REAL	CV3 rate of change limit, in percent per second. Rate of change limiting is only used when in Auto mode or in Manual mode if CVManLimiting is TRUE. A value of zero disables CV3 ROC limiting. If value of CV3ROCLimit < 0, set bit in Status and disable CV3 ROC limiting.	Valid = 0.0 to maximum positive float Default = 0.0
CV1ROCNegLimit	REAL	CV1 rate of change limit, in percent per second. Rate of change limiting is only used when in Auto mode or in Manual mode if CVManLimiting is TRUE. A value of zero disables CV1 ROC limiting. If value of CV1ROCLimit < 0, set bit in Status and disable CV1 ROC limiting.	Valid = 0.0 to maximum positive float Default = 0.0
CV2ROCNegLimit	REAL	CV2 rate of change limit, in percent per second. Rate of change limiting is only used when in Auto mode or in Manual mode if CVManLimiting is TRUE. A value of zero disables CV2 ROC limiting. If value of CV2ROCLimit < 0, set bit in Status and disable CV2 ROC limiting.	Valid = 0.0 to maximum positive float Default = 0.0
CV3ROCNegLimit	REAL	CV3 rate of change limit, in percent per second.	Valid = 0.0 to maximum positive float

Input Parameters	Data Type	Description	Values
		Rate of change limiting is only used when in Auto mode or in Manual mode if CVManLimiting is TRUE. A value of zero disables CV3 ROC limiting. If value of CV3ROCLimit < 0, set bit in Status and disable CV3 ROC limiting.	Default = 0.0
CV1HandFB	REAL	CV1 HandFeedback value. CV1 set to this value when in Hand mode and CV1HandFBFault is FALSE (good health). This value would typically come from the output of a field mounted hand/auto station and would be used to generate a bumpless transfer out of Hand mode. If value of CV1HandFB < 0 or > 100, set unique Status bit and limit value used for CV1.	Valid = 0.0 through 100.0 Default = 0.0
CV2HandFB	REAL	CV2 HandFeedback value. CV2 set to this value when in Hand mode and CV2HandFBFault is FALSE (good health). This value would typically come from the output of a field mounted hand/auto station and would be used to generate a bumpless transfer out of Hand mode. If value of CV2HandFB < 0 or > 100, set unique Status bit and limit value used for CV2.	Valid = 0.0 through 100.0 Default = 0.0
CV3HandFB	REAL	CV3 HandFeedback value. CV3 set to this value when in Hand mode and CV3HandFBFault is FALSE (good health). This value would typically come from the output of a field mounted hand/auto	Valid = 0.0 through 100.0 Default = 0.0

Input Parameters	Data Type	Description	Values
		station and would be used to generate a bumpless transfer out of Hand mode. If value of CV3HandFB < 0 or > 100, set unique Status bit and limit value used for CV3.	
CV1HandFBFault	BOOL	CV1HandFB value bad health indicator. If the CV1HandFB value is read from an analog input, then CV1HandFBFault will normally be controlled by the status of the analog input channel. If CV1HandFBFault is TRUE, it indicates an error on the input module, set bit in Status. FALSE = Good Health	Default = FALSE
CV2HandFBFault	BOOL	CV2HandFB value bad health indicator. If the CV2HandFB value is read from an analog input, then CV2HandFBFault will normally be controlled by the status of the analog input channel. If CV2HandFBFault is TRUE, it indicates an error on the input module, set bit in Status. FALSE = Good Health	Default = FALSE
CV3HANDFBFault	BOOL	CV3HandFB value bad health indicator. If the CV3HandFB value is read from an analog input, then CV3HandFBFault will normally be controlled by the status of the analog input channel. If CV3HandFBFault is TRUE, it indicates an error on the input module, set bit in Status.	Default = FALSE

Input Parameters	Data Type	Description	Values
		FALSE = Good Health	
CV1Target	REAL	Target value for control variable output 1.	Valid = 0.0 through 100.0 Default = 0.0
CV2Target	REAL	Target value for control variable output 2.	Valid = 0.0 through 100.0 Default = 0.0
CV3Target	REAL	Target value for control variable output 3.	Valid = 0.0 through 100.0 Default = 0.0
CV1WindupHIn	BOOL	CV1Windup high request. When TRUE, CV1 will not be allowed to increase in value. This signal will typically be the CV1WindupHOut output from a secondary loop.	Default = FALSE
CV2WindupHIn	BOOL	CV2Windup high request. When TRUE, CV2 will not be allowed to increase in value. This signal will typically be the CV2WindupHOut output from a secondary loop.	Default = FALSE
CV3WindupHIn	BOOL	CV3Windup high request. When TRUE, CV3 will not be allowed to increase in value. This signal will typically be the CV3WindupHOut output from a secondary loop.	Default = FALSE
CV1WindupLIn	BOOL	CV1 Windup low request. When TRUE, CV1 will not be allowed to decrease in value. This signal will typically be the CV1WindupLOut output from a secondary loop.	Default = FALSE
CV2WindupLIn	BOOL	CV2 Windup low request. When TRUE, CV2 will not be allowed to decrease in value. This signal will typically be the CV2WindupLOut output from a secondary loop.	Default = FALSE

Input Parameters	Data Type	Description	Values
CV3WindupLIn	BOOL	CV3 Windup low request. When TRUE, CV3 will not be allowed to decrease in value. This signal will typically be the CV3WindupLOut output from a secondary loop.	Default = FALSE
GainEUSpan	BOOL	ModelGain units in EU or as % of span.	Default = FALSE FALSE = Gain in % of span
CV1PV1ProcessGainSign	BOOL	Used only for Autotuning. Sign of the process gain (Delta PV1/Delta CV1). • Set to indicate a negative process gain (increase in output causes a decrease in PV1). • Reset to indicate a positive process gain (increase in output causes an increase in P1V).	Default = FALSE
CV2PV1ProcessGainSign	BOOL	Used only for Autotuning. Sign of the process gain (Delta PV1/Delta CV2). • Set to indicate a negative process gain (increase in output causes a decrease in PV1). • Reset to indicate a positive process gain (increase in output causes an increase in PV1).	Default = FALSE
CV3PV1ProcessGainSign	BOOL	Used only for Autotuning. Sign of the process gain (Delta PV1/Delta CV3). • Set to indicate a negative process gain (increase in output causes a decrease in PV1).	Default = FALSE

Input Parameters	Data Type	Description	Values
		• Reset to indicate a positive process gain (increase in output causes an increase in PV1).	
CV1PV2ProcessGainSign	BOOL	Used only for Autotuning. Sign of the process gain (Delta PV2/Delta CV1). • Set to indicate a negative process gain (increase in output causes a decrease in PV2). • Reset to indicate a positive process gain (increase in output causes an increase in PV2).	Default = FALSE
CV1PV2ProcessGainSign	BOOL	Used only for Autotuning. Sign of the process gain (Delta PV2/Delta CV2). • Set to indicate a negative process gain (increase in output causes a decrease in PV2). • Reset to indicate a positive process gain (increase in output causes an increase in PV2).	Default = FALSE
CV1PV2ProcessGainSign	BOOL	Used only for Autotuning. Sign of the process gain (Delta PV2/Delta CV3). • Set to indicate a negative process gain (increase in output causes a decrease in PV2). • Reset to indicate a positive process gain (increase in output causes an increase in PV2).	Default = FALSE

Input Parameters	Data Type	Description	Values
ProcessType	DINT	Process type selection for both PV1 and PV2 (1=Integrating, 0=non-integrating)	Default = 0
CV1PV1ModelGain	REAL	The internal model gain parameter for CV1 - PV1. Enter a positive or negative gain depending on process direction. If CV1PV1ModelGain = INF or NAN, set bit in Status.	Valid = maximum negative float –> maximum positive float Default = 0.0
CV2PV1ModelGain	REAL	The internal model gain parameter for CV2 - PV1. Enter a positive or negative gain depending on process direction. If CV2PV1ModelGain = INF or NAN, set bit in Status.	Valid = maximum negative float –> maximum positive float Default = 0.0
CV3PV1ModelGain	REAL	The internal model gain parameter for CV3 - PV1. Enter a positive or negative gain depending on process direction. If CV3PV1ModelGain = INF or NAN, set bit in Status.	Valid = maximum negative float –> maximum positive float Default = 0.0
CV1PV2ModelGain	REAL	The internal model gain parameter for CV1 - PV2. Enter a positive or negative gain depending on process direction. If CV1PV2ModelGain = INF or NAN, set bit in Status.	Valid = maximum negative float –> maximum positive float Default = 0.0
CV2PV2ModelGain	REAL	The internal model gain parameter for CV2 - PV2. Enter a positive or negative gain depending on process direction. If CV2PV2ModelGain = INF or NAN, set bit in Status.	Valid = maximum negative float –> maximum positive float Default = 0.0
CV3PV2ModelGain	REAL	The internal model gain parameter for CV3 - PV2. Enter a positive or negative gain depending on process direction.	Valid = maximum negative float –> maximum positive float Default = 0.0

Input Parameters	Data Type	Description	Values
		If CV3PV2ModelGain = INF or NAN, set bit in Status.	
CV1PV1ModelTC	REAL	The internal model time constant for CV1 - PV1 in seconds.	Valid = 0.0 to maximum positive float Default = 0.0
CV2PV1ModelTC	REAL	The internal model time constant for CV2 - PV1 in seconds.	Valid = 0.0 to maximum positive float Default = 0.0
CV3PV1ModelTC	REAL	The internal model time constant for CV3 - PV1 in seconds.	Valid = 0.0 to maximum positive float Default = 0.0
CV1PV2ModelTC	REAL	The internal model time constant for CV1 - PV2 in seconds.	Valid = 0.0 to maximum positive float Default = 0.0
CV2PV2ModelTC	REAL	The internal model time constant for CV2 - PV2 in seconds.	Valid = 0.0 to maximum positive float Default = 0.0
CV3PV2ModelTC	REAL	The internal model time constant for CV3 - PV2 in seconds.	Valid = 0.0 to maximum positive float Default = 0.0
CV1PV1ModelDT	REAL	The internal model deadtime for CV1 - PV1 in seconds.	Valid = 0.0 to maximum positive float Default = 0.0
CV2PV1ModelDT	REAL	The internal model deadtime for CV2 - PV1 in seconds.	Valid = 0.0 to maximum positive float Default = 0.0
CV3PV1ModelDT	REAL	The internal model deadtime for CV3 - PV1 in seconds.	Valid = 0.0 to maximum positive float Default = 0.0
CV1PV2ModelDT	REAL	The internal model deadtime for CV1 - PV2 in seconds.	Valid = 0.0 to maximum positive float Default = 0.0
CV2PV2ModelDT	REAL	The internal model deadtime for CV2 - PV2 in seconds.	Valid = 0.0 to maximum positive float Default = 0.0

Input Parameters	Data Type	Description	Values
CV3PV2ModelDT	REAL	The internal model deadtime for CV3 - PV2 in seconds.	Valid = 0.0 to maximum positive float Default = 0.0
CV1PV1RespTC	REAL	The tuning parameter that determines the speed of the control variable action for CV1 - PV1 in seconds.	Valid = 0.0 to maximum positive float Default = 0.0
CV2PV1RespTC	REAL	The tuning parameter that determines the speed of the control variable action for CV2 - PV1 in seconds.	Valid = 0.0 to maximum positive float Default = 0.0
CV3PV1RespTC	REAL	The tuning parameter that determines the speed of the control variable action for CV3 - PV1 in seconds.	Valid = 0.0 to maximum positive float Default = 0.0
CV1PV2RespTC	REAL	The tuning parameter that determines the speed of the control variable action for CV1 - PV2 in seconds.	Valid = 0.0 to maximum positive float Default = 0.0
CV2PV2RespTC	REAL	The tuning parameter that determines the speed of the control variable action for CV2 - PV2 in seconds.	Valid = 0.0 to maximum positive float Default = 0.0
CV3PV2RespTC	REAL	The tuning parameter that determines the speed of the control variable action for CV3 - PV2 in seconds.	Valid = 0.0 to maximum positive float Default = 0.0
PV1Act1stCV	DINT	The first CV to act to compensate for PV1-SP1 deviation. 1=CV1, 2=CV2, 3=CV3	Valid = 1-3 Default = 1
PV1Act2ndCV	DINT	The second CV to act to compensate for PV1-SP1 deviation. 1=CV1, 2=CV2, 3=CV3	Valid = 1-3 Default = 2
PV1Act3rdCV	DINT	The third CV to act to compensate for PV1-SP1 deviation. 1=CV1, 2=CV2, 3=CV3	Valid = 1-3 Default = 3

Input Parameters	Data Type	Description	Values
PV2Act1stCV	DINT	The first CV to act to compensate for PV2-SP2 deviation. 1=CV1, 2=CV2, 3=CV3	Valid = 1-3 Default = 1
PV2Act2ndCV	DINT	The second CV to act to compensate for PV2-SP2 deviation. 1=CV1, 2=CV2, 3=CV3	Valid = 1-3 Default = 2
PV2Act3rdCV	DINT	The third CV to act to compensate for PV2-SP2 deviation. 1=CV1, 2=CV2, 3=CV3	Valid = 1-3 Default = 3
TargetCV	DINT	The CV to be driven to its target value. 1=CV1, 2=CV2, 3=CV3	Valid = 1-3 Default = 3
TargetRespTC	REAL	Determines the speed with which the control variables approach the target values.	Valid = 0.0 to maximum positive float Default = 0.0
PVTracking	BOOL	SP track PV request. Set TRUE to enable SP to track PV. Ignored when in Auto modes. SP will only track PV when all three outputs are in manual. As soon as any output returns to Auto, PVTracking stops.	Default = FALSE
ManualAfterInit	BOOL	Manual mode after initialization request. • When TRUE, the appropriate CV(n), where (n) can be 1, 2, or 3, will be placed in Manual mode when CV(n)Initializing is set TRUE unless the current mode is Override or Hand. • When ManualAfterInit is FALSE, the CV(n) mode will not be changed.	Default = FALSE

Input Parameters	Data Type	Description	Values
ProgProgReq	BOOL	Program Program Request. • Set TRUE by the user program to request Program control. Ignored if ProgOperReq is TRUE. Holding this TRUE and ProgOperReq FALSE can be used to lock the function block into program control. • When ProgValueReset is TRUE, the function block resets the input to FALSE.	Default = FALSE
ProgOperReq	BOOL	Program Operator Request. • Set TRUE by the user program to request Operator control. Holding this TRUE can be used to lock the function block into operator control. • When ProgValueReset is TRUE, the function block resets the input to FALSE.	Default = FALSE
ProgCV1AutoReq	BOOL	Program-Auto mode request for CV1. • Set TRUE by the user program to request Auto mode. • When ProgValueReset is TRUE, the function block resets the input to FALSE.	Default = FALSE

Input Parameters	Data Type	Description	Values
ProgCV2AutoReq	BOOL	Program-Auto mode request for CV2. • Set TRUE by the user program to request Auto mode. • When ProgValueReset is TRUE, the function block resets the input to FALSE.	Default = FALSE
ProgCV3AutoReq	BOOL	Program-Auto mode request for CV3. • Set TRUE by the user program to request Auto mode. • When ProgValueReset is TRUE, the function block resets the input to FALSE.	Default = FALSE
ProgCV1ManualReq	BOOL	Program-Manual mode request for CV1. • Set TRUE by the user program to request Manual mode. • When ProgValueReset is TRUE, the function block resets the input to FALSE.	Default = FALSE
ProgCV2ManualReq	BOOL	Program-Manual mode request for CV2. • Set TRUE by the user program to request Manual mode. • When ProgValueReset is TRUE, the function block resets the input to FALSE.	Default = FALSE

Input Parameters	Data Type	Description	Values
ProgCV3ManualReq	BOOL	Program-Manual mode request for CV3. • Set TRUE by the user program to request Manual mode. • When ProgValueReset is TRUE, the function block resets the input to FALSE.	Default = FALSE
ProgCV1OverrideReq	BOOL	Program-Override mode request for CV1. • Set TRUE by the user program to request Override mode. • When ProgValueReset is TRUE, the function block resets the input to FALSE.	Default = FALSE
ProgCV2OverrideReq	BOOL	Program-Override mode request for CV2. • Set TRUE by the user program to request Override mode. • When ProgValueReset is TRUE, the function block resets the input to FALSE.	Default = FALSE
ProgCV3OverrideReq	BOOL	Program-Override mode request for CV3. • Set TRUE by the user program to request Override mode. • When ProgValueReset is TRUE, the function block resets the input to FALSE.	Default = FALSE
ProgCV1HandReq	BOOL	Program-Hand mode request for CV1. • Set TRUE by the user program to request Hand mode. This value will usually be read as	Default = FALSE

Input Parameters	Data Type	Description	Values
		a digital input from a hand/auto station. • When ProgValueReset is TRUE, the function block resets the input to FALSE.	
ProgCV2HandReq	BOOL	Program-Hand mode request for CV2. • Set TRUE by the user program to request Hand mode. This value will usually be read as a digital input from a hand/auto station. • When ProgValueReset is TRUE, the function block resets the input to FALSE.	Default = FALSE
ProgCV3HandReq	BOOL	Program-Hand mode request for CV3. • Set TRUE by the user program to request Hand mode. This value will usually be read as a digital input from a hand/auto station. • When ProgValueReset is TRUE, the function block resets the input to FALSE.	Default = FALSE
OperProgReq	BOOL	Operator Program Request. • Set TRUE by the operator interface to request Program control. The function block resets this parameter to FALSE.	Default = FALSE
OperOperReq	BOOL	Operator Operator Request. • Set TRUE by the operator interface to request Operator	Default = FALSE

Input Parameters	Data Type	Description	Values
		control. The function block resets this parameter to FALSE.	
OperCV1AutoReq	BOOL	Operator-Auto mode request for CV1. Set TRUE by the operator interface to request Auto mode. The function block resets this parameter to FALSE.	Default = FALSE
OperCV2AutoReq	BOOL	Operator-Auto mode request for CV2. Set TRUE by the operator interface to request Auto mode. The function block resets this parameter to FALSE.	Default = FALSE
OperCV3AutoReq	BOOL	Operator-Auto mode request for CV3. Set TRUE by the operator interface to request Auto mode. The function block resets this parameter to FALSE.	Default = FALSE
OperCV1ManualReq	BOOL	Operator-Manual mode request for CV1. Set TRUE by the operator interface to request Manual mode. The function block sets this parameter to FALSE.	Default = FALSE
OperCV2ManualReq	BOOL	Operator-Manual mode request for CV2. Set TRUE by the operator interface to request Manual mode. The function block sets this parameter to FALSE.	Default = FALSE
OperCV3ManualReq	BOOL	Operator-Manual mode request for CV3. Set TRUE by the operator interface to request Manual mode. The function	Default = FALSE

Input Parameters	Data Type	Description	Values
		block sets this parameter to FALSE.	
ProgValueReset	BOOL	Reset Program control values. When TRUE, the Prog_xxx_Req inputs are reset to FALSE. When TRUE and Program control, set SP(x)Prog = SP(x) and CV(y)Prog = CV(y), where x = 1,2 and y = 1,2,3	Default = FALSE
TimingMode	DINT	Selects Time Base Execution mode. Value/Description 0 = Periodic mode 1 = Oversample mode 2 = Real time sampling mode For more information about timing modes, see Function Block Attributes.	Valid = 0 through 2 Default = 0
OverSampleDT	REAL	Execution time for Oversample mode.	Valid = 0 to max. TON_Timer elapsed time (4194.303 seconds) Default = 0
RTSTime	DINT	Module update period for Real Time Sampling mode.	Valid = 0 through 32,767 1 count = 1 ms
RTSTimeStamp	DINT	Module time stamp value for Real Time Sampling mode.	Valid = 0 through 32,767 (wraps from 32,767...0) 1 count = 1 ms
PV1TuneLimit	REAL	PV1 tuning limit scaled in the PV1 units. When Autotune is running and predicted PV1 exceeds this limit, the tuning will be aborted.	Valid = any float Default=0
PV2TuneLimit	REAL	PV2 tuning limit scaled in the PV2 units. When Autotune is running and predicted PV2 exceeds	Valid = any float Default=0

Input Parameters	Data Type	Description	Values
		this limit, the tuning will be aborted.	
PV1AtuneTimeLimit	REAL	Maximum time in minutes for PV1 autotune to complete following the CV1 step change. When PV1 autotune exceeds this time, tuning will be aborted.	Valid range: any float > 0. Default = 60 minutes
PV2AtuneTimeLimit	REAL	Maximum time in minutes for PV2 autotune to complete following the CV2 step change. When PV2 autotune exceeds this time, tuning will be aborted.	Valid range: any float > 0. Default = 60 minutes
PV1NoiseLevel	DINT	An estimate of the noise level expected on the PV1 to compensate for it during tuning. The selections are: 0=low, 1=medium, 2=high	Range: 0 through 2 Default=1
PV2NoiseLevel	DINT	An estimate of the noise level expected on the PV2 to compensate for it during tuning. The selections are: 0=low, 1=medium, 2=high	Range: 0 through 2 Default=1
CV1StepSize	REAL	CV1 step size in percent for the tuning step test. Step size is directly added to CV1 subject to high/low limiting.	Range: -100% ... 100% Default=10%
CV2StepSize	REAL	CV2 step size in percent for the tuning step test. Step size is directly added to CV2 subject to high/low limiting.	Range: -100% ... 100% Default=10%
CV3StepSize	REAL	CV3 step size in percent for the tuning step test. Step size is directly added to CV3 subject to high/low limiting.	Range: -100% ... 100% Default=10%

Input Parameters	Data Type	Description	Values
CV1PV1ResponseSpeed	DINT	Desired speed of closed loop response for CV1 - PV1. • Slow response: ResponseSpeed=0 • Medium response: ResponseSpeed=1 • Fast response: ResponseSpeed=2 If ResponseSpeed is less than 0, Slow response is used. If ResponseSpeed is greater than 2, Fast response is used.	Range: 0...2 Default=1
CV2PV1ResponseSpeed	DINT	Desired speed of closed loop response for CV2 PV1. • Slow response: ResponseSpeed=0 • Medium response: ResponseSpeed=1 • Fast response: ResponseSpeed=2 If ResponseSpeed is less than 0, Slow response is used. If ResponseSpeed is greater than 2, Fast response is used.	Range: 0...2 Default=1
CV3PV1ResponseSpeed	DINT	Desired speed of closed loop response for CV3 PV1. • Slow response: ResponseSpeed=0 • Medium response: ResponseSpeed=1 • Fast response: ResponseSpeed=2 If ResponseSpeed is less than 0, Slow response is used. If ResponseSpeed is greater than 2, Fast response is used.	Range: 0 through 2 Default=1

Input Parameters	Data Type	Description	Values
CV1PV2ResponseSpeed	DINT	Desired speed of closed loop response for CV1 PV2. • Slow response: ResponseSpeed=0 • Medium response: ResponseSpeed=1 • Fast response: ResponseSpeed=2 If ResponseSpeed is less than 0, Slow response is used. If ResponseSpeed is greater than 2, Fast response is used.	Range: 0 through 2 Default=1
CV2PV2ResponseSpeed	DINT	Desired speed of closed loop response for CV2 PV2. • Slow response: ResponseSpeed=0 • Medium response: ResponseSpeed=1 • Fast response: ResponseSpeed=2 If ResponseSpeed is less than 0, Slow response is used. If ResponseSpeed is greater than 2, Fast response is used.	Range: 0 through 2 Default=1
CV3PV2ResponseSpeed	DINT	Desired speed of closed loop response for CV3 PV2. • Slow response: ResponseSpeed=0 • Medium response: ResponseSpeed=1 • Fast response: ResponseSpeed=2 If ResponseSpeed is less than 0, Slow response is used. If ResponseSpeed is greater than 2, Fast response is used.	Range: 0 through 2 Default=1
CV1PV1ModelInit	BOOL	Internal model initialization switch for CV1 - PV1. Refer to Function Block Attributes.	Default = FALSE

Input Parameters	Data Type	Description	Values
CV2PV1ModelInit	BOOL	Internal model initialization switch for CV2 - PV1. Refer to Function Block Attributes.	Default = FALSE
CV3PV1ModelInit	BOOL	Internal model initialization switch for CV3 - PV1. Refer to Function Block Attributes.	Default = FALSE
CV1PV2ModelInit	BOOL	Internal model initialization switch for CV1 - PV2. Refer to Function Block Attributes.	Default = FALSE
CV2PV2ModelInit	BOOL	Internal model initialization switch for CV2 - PV2. Refer to Function Block Attributes.	Default = FALSE
CV3PV2ModelInit	BOOL	Internal model initialization switch for CV3 - PV2. Refer to Function Block Attributes.	Default = FALSE
PV1Factor	REAL	Non-integrating model approximation factor for PV1. Only used for integrating process types.	Default = 100
PV2Factor		Non-integrating model approximation factor for PV2. Only used for integrating process types.	Default = 100
AtuneCV1Start	BOOL	Start Autotune request for CV1. Set True to initiate auto tuning of the CV1 output for both PV1 and PV2. Ignored when CV1 is not in Manual mode. The function block resets the input to FALSE.	Default = FALSE
AtuneCV2Start	BOOL	Start Autotune request for CV2. Set True to initiate auto tuning of the CV2 output for both PV1 and PV2. Ignored when CV2 is not in Manual mode. The	Default = FALSE

Input Parameters	Data Type	Description	Values
		function block resets the input to FALSE.	
AtuneCV3Start	BOOL	Start Autotune request for CV3. Set True to initiate auto tuning of the CV3 output for both PV1 and PV2. Ignored when CV3 is not in Manual mode. The function block resets the input to FALSE.	Default = FALSE
AtuneCV1PV1UseModel	BOOL	Use Autotune model request for CV1 - PV1. Set True to replace the current model parameters with the calculated Autotune model parameters. The function block resets the input parameter to FALSE.	Default = FALSE
AtuneCV2PV1UseModel	BOOL	Use Autotune model request for CV2 - PV1. Set True to replace the current model parameters with the calculated Autotune model parameters. The function block resets the input parameter to FALSE.	Default = FALSE
AtuneCV3PV1UseModel	BOOL	Use Autotune model request for CV3 - PV1. Set True to replace the current model parameters with the calculated Autotune model parameters. The function block resets the input parameter to FALSE.	Default = FALSE
AtuneCV1PV2UseModel	BOOL	Use Autotune model request for CV1 - PV2. Set True to replace the current model parameters with the calculated Autotune model parameters. The function block resets the input parameter to FALSE.	Default = FALSE
AtuneCV2PV2UseModel	BOOL	Use Autotune model request for CV2 - PV2. Set True to replace the current model parameters with	Default = FALSE

Input Parameters	Data Type	Description	Values
		the calculated Autotune model parameters. The function block resets the input parameter to FALSE.	
AtuneCV3PV2UseModel	BOOL	Use Autotune model request for CV3 - PV2. Set True to replace the current model parameters with the calculated Autotune model parameters. The function block resets the input parameter to FALSE.	Default = FALSE
AtuneCV1Abort	BOOL	Abort Autotune request for CV1. Set True to abort the auto tuning of CV1 output for both PV1 and PV2. The function block resets input parameter to FALSE.	Default = FALSE
AtuneCV2Abort	BOOL	Abort Autotune request for CV2. Set True to abort the auto tuning of CV2 output or both PV1 and PV2. The function block resets input parameter to FALSE.	Default = FALSE
AtuneCV3Abort	BOOL	Abort Autotune request for CV3. Set True to abort the auto tuning of CV3 output or both PV1 and PV2. The function block resets input parameter to FALSE.	Default = FALSE

The following table describes the output parameters in the MMC function block.

Output Parameters	Data Type	Description	Values
EnableOut	BOOL	Indicates if instruction is enabled. Cleared to false if any one of CV1EU, CV2EU or CV3EU overflows.	
CV1EU	REAL	Scaled control variable output for CV1. Scaled by using CV1EUMax and CV1EUMin, where CV1EUMax corresponds to 100% and CV1EUMin corresponds to 0%. This output is	

Output Parameters	Data Type	Description	Values
		typically used to control an analog output module or a secondary loop. CV1EU = (CV1 * CV1EUSpan / 100) + CV1EUMin CV1EU span calculation: CV1EUSpan = (CV1EUMax – CV1EUMin)	
CV2EU	REAL	Scaled control variable output for CV2. Scaled by using CV2EUMax and CV2EUMin, where CV2EUMax corresponds to 100% and CV2EUMin corresponds to 0%. This output is typically used to control an analog output module or a secondary loop. CV2EU = (CV2 * CV2EUSpan / 100) + CV2EUMin CV2EU span calculation: CV2EUSpan = (CV2EUMax – CV2EUMin)	
CV3EU	REAL	Scaled control variable output for CV3. Scaled by using CV3EUMax and CV3EUMin, where CV3EUMax corresponds to 100% and CV3EUMin corresponds to 0%. This output is typically used to control an analog output module or a secondary loop. CV3EU = (CV3 * CV3EUSpan / 100) + CV3EUMin CV3EU span calculation: CV3EUSpan = (CV3EUMax – CV3EUMin)	
CV1	REAL	Control variable output for CV1. This value will always be expressed as 0...100%. CV1 is limited by CV1HLimit	

Output Parameters	Data Type	Description	Values
		and CV1LLimit when in Auto mode or in Manual mode if CVManLimiting is TRUE; otherwise limited by 0 and 100%.	
CV2	REAL	Control variable output for CV2. This value will always be expressed as 0...100%. CV2 is limited by CV2HLimit and CV2LLimit when in Auto mode or in Manual mode if CVManLimiting is TRUE; otherwise limited by 0 and 100%.	
CV3	REAL	Control variable output for CV3. This value will always be expressed as 0...100%. CV3 is limited by CV3HLimit and CV3LLimit when in Auto mode or in Manual mode if CVManLimiting is TRUE; otherwise limited by 0 and 100%.	
CV1Initializing	BOOL	Initialization mode indicator for CV1. Set TRUE when CV1InitReq, function blockFirstScan or OLCFirstRun, are TRUE, or on a TRUE to FALSE transition of CV1Fault (bad to good). CV1Initializing is set FALSE after the function block has been initialized and CV1InitReq is no longer TRUE.	
CV2nitializing	BOOL	Initialization mode indicator for CV2. Set TRUE when CV2InitReq, function blockFirstScan or OLCFirstRun, are TRUE, or on a TRUE to FALSE transition of CV2Fault (bad to good). CV2Initializing is set FALSE after the function block has been	

Output Parameters	Data Type	Description	Values
		initialized and CV2InitReq is no longer TRUE.	
CV3nitializing	BOOL	Initialization mode indicator for CV3. Set TRUE when CV3InitReq, function blockFirstScan or OLCFirstRun, are TRUE, or on a TRUE to FALSE transition of CV3Fault (bad to good). CV3Initializing is set FALSE after the function block has been initialized and CV3InitReq is no longer TRUE.	
CV1HAlarm	BOOL	CV1 high alarm indicator. TRUE when the calculated value for CV1 > 100 or CV1HLimit.	
CV2HAlarm	BOOL	CV2 high alarm indicator. TRUE when the calculated value for CV2 > 100 or CV2HLimit.	
CV3HAlarm	BOOL	CV3 high alarm indicator. TRUE when the calculated value for CV3 > 100 or CV3HLimit.	
CV1LAlarm	BOOL	CV1 low alarm indicator. TRUE when the calculated value for CV1 < 0 or CV1LLimit.	
CV2LAlarm	BOOL	CV2 low alarm indicator. TRUE when the calculated value for CV2 < 0 or CV2LLimit.	
CV3LAlarm	BOOL	CV3 low alarm indicator. TRUE when the calculated value for CV3 < 0 or CV3LLimit.	
CV1ROCPosAlarm	BOOL	CV1 rate of change alarm indicator. TRUE when the calculated rate of change for CV1 exceeds CV1ROCPosLimit.	

Output Parameters	Data Type	Description	Values
CV2ROCPosAlarm	BOOL	CV2 rate of change alarm indicator. TRUE when the calculated rate of change for CV2 exceeds CV2ROCPosLimit.	
CV3ROCPosAlarm	BOOL	CV3 rate of change alarm indicator. TRUE when the calculated rate of change for CV3 exceeds CV3ROCPosLimit.	
CV1ROCNegAlarm	BOOL	CV1 rate of change alarm indicator. TRUE when the calculated rate of change for CV1 exceeds CV1ROCNegLimit.	
CV2ROCNegAlarm	BOOL	CV2 rate of change alarm indicator. TRUE when the calculated rate of change for CV2 exceeds CV2ROCNegLimit.	
CV3ROCNegAlarm	BOOL	CV3 rate of change alarm indicator. TRUE when the calculated rate of change for CV3 exceeds CV3ROCNegLimit.	
SP1	REAL	Current setpoint 1 value. The value of SP1 is used to control CV when in the Auto or the PV1 Tracking mode, scaled in PV1 units.	
SP2	REAL	Current setpoint 2 value. The value of SP2 is used to control CV when in the Auto or the PV2 Tracking mode, scaled in PV2 units.	
SP1Percent	REAL	The value of SP1 expressed in percent of span of PV1. SP1Percent = ((SP1 – PV1EUMin) * 100) / PV1Span where PV1Span = PV1EUMax – PV1EUMin	

Output Parameters	Data Type	Description	Values
SP2Percent		The value of SP2 expressed in percent of span of PV2. SP2Percent = ((SP2 – PV2EUMin) * 100) / PV2Span where PV2Span = PV2EUMax – PV1EUMin	
SP1HAlarm	BOOL	SP1 high alarm indicator. TRUE when the SP1 >= SP1HLimit.	
SP2HAlarm	BOOL	SP2 high alarm indicator. TRUE when the SP2 >= SP2HLimit.	
SP1LAlarm	BOOL	SP1 low alarm indicator. TRUE when the SP1 <= SP1LLimit.	
SP2LAlarm	BOOL	SP2 low alarm indicator. TRUE when the SP2 <= SP2LLimit.	
PV1Percent	REAL	PV1 expressed in percent of span. PV1Percent = ((PV1 – PV1EUMin) * 100) / PV1Span PV1 Span calculation: PV1Span = (PV1EUMax - PV1EUMin)	
PV2Percent	REAL	PV2 expressed in percent of span. PV2Percent = ((PV2 – PV2EUMin) * 100) / PV2Span PV2 Span calculation: PV2Span = (PV2EUMax – PV2EUMin)	
E1	REAL	Process1 error. Difference between SP1 and PV1, scaled in PV1 units.	
E2	REAL	Process2 error. Difference between SP2 and PV2, scaled in PV2 units.	

Output Parameters	Data Type	Description	Values
E1Percent	REAL	The error expressed as a percent of span for process 1.	
E2Percent	REAL	The error expressed as a percent of span for process 2.	
CV1WindupHOut	BOOL	CV1 Windup high indicator. CV1WindupHOut is set to true when • SP1HAlarm or SP2HAlarm is true or • ModelGain is positive and CV1HAlarm is true or • ModelGain is negative and CV1LAlarm is true. This signal will typically be used by the WindupHIn input to limit the windup of the CV1 output on a primary loop.	
CV2WindupHOut	BOOL	CV2 Windup high indicator. CV2WindupHOut is set to true when • SP1HAlarm or SP2HAlarm is true or • ModelGain is positive and CV2HAlarm is true or • ModelGain is negative and CV2LAlarm is true. This signal will typically be used by the WindupHIn input to limit the windup of the CV2 output on a primary loop.	
CV3WindupHOut	BOOL	CV3 Windup high indicator.	

Output Parameters	Data Type	Description	Values
		CV3WindupHOut is set to true when • SP1HAlarm or SP2HAlarm is true or • ModelGain is positive and CV3HAlarm is true or • ModelGain is negative and CV3LAlarm is true. This signal will typically be used by the WindupHIn input to limit the windup of the CV3 output on a primary loop.	
CV1WindupLOut	BOOL	CV1 Windup low indicator. CV1WindupLOut is set to true when • SP1LAlarm or SP2LAlarm is true or • ModelGain is positive and CV1LAlarm is true or • ModelGain is negative and CV1HAlarm is true. This signal will typically be used by the WindupLIn input to limit the windup of the CV1 output on a primary loop.	
CV2WindupLOut	BOOL	CV2 Windup low indicator. CV2WindupLOut is set to true when • SP1LAlarm or SP2LAlarm is true or • ModelGain is positive and CV2LAlarm is true or • ModelGain is negative and CV2HAlarm is true. This signal will typically be used by the WindupLIn input to limit the windup	

Output Parameters	Data Type	Description	Values
		of the CV2 output on a primary loop.	
CV3WindupLOut	BOOL	CV3 Windup low indicator. CV3WindupLOut is set to true when • SP1LAlarm or SP2LAlarm is true or • ModelGain is positive and CV3LAlarm is true or • ModelGain is negative and CV3HAlarm is true. This signal will typically be used by the WindupLIn input to limit the windup of the CV3 output on a primary loop.	
ProgOper	BOOL	Program/Operator control indicator. TRUE when in Program control. FALSE when in Operator control.	
CV1Auto	BOOL	Auto mode indicator for CV1. TRUE when CV1 in the Auto mode.	
CV2Auto	BOOL	Auto mode indicator for CV2. TRUE when CV2 in the Auto mode.	
CV2Auto	BOOL	Auto mode indicator for CV3. TRUE when CV3 in the Auto mode.	
CV1Manual	BOOL	Manual mode indicator for CV1. TRUE when CV1 in the Manual mode.	
CV2Manual	BOOL	Manual mode indicator for CV2. TRUE when CV2 in the Manual mode.	
CV3Manual	BOOL	Manual mode indicator for CV3. TRUE when CV3 in the Manual mode.	

Output Parameters	Data Type	Description	Values
CV1Override	BOOL	Override mode indicator for CV1. TRUE when CV1 in the Override mode.	
CV2Override	BOOL	Override mode indicator for CV2. TRUE when CV2 in the Override mode.	
CV3Override	BOOL	Override mode indicator for CV3. TRUE when CV3 in the Override mode.	
CV1Hand	BOOL	Hand mode indicator for CV1. TRUE when CV1 in the Hand mode.	
CV2Hand	BOOL	Hand mode indicator for CV2. TRUE when CV2 in the Hand mode.	
CV3Hand	BOOL	Hand mode indicator for CV3. TRUE when CV3 in the Hand mode.	
DeltaT	REAL	Elapsed time between updates in seconds.	
CV1StepSizeUsed	REAL	Actual CV1 step size used for tuning.	
CV2StepSizeUsed	REAL	Actual CV2 step size used for tuning.	
CV3StepSizeUsed	REAL	Actual CV3 step size used for tuning.	
CV1PV1GainTuned	REAL	The calculated value of the internal model gain for CV1 - PV1 after tuning is completed.	
CV2PV1GainTuned	REAL	The calculated value of the internal model gain for CV2 - PV1 after tuning is completed.	
CV3PV1GainTuned	REAL	The calculated value of the internal model gain for CV3 - PV1 after tuning is completed.	

Output Parameters	Data Type	Description	Values
CV1PV2GainTuned	REA:	The calculated value of the internal model gain for CV1 - PV2 after tuning is completed.	
CV2PV2GainTuned	REAL	The calculated value of the internal model gain for CV2 - PV2 after tuning is completed.	
CV3PV2GainTuned	REA:	The calculated value of the internal model gain for CV3 - PV2 after tuning is completed.	
CV1PV1TCTuned	REAL	The calculated value of the internal model time constant for CV1 - PV1 after tuning is completed.	
CV2PV1TCTuned	REAL	The calculated value of the internal model time constant for CV2 - PV1 after tuning is completed.	
CV3PV1TCTuned	REAL	The calculated value of the internal model time constant for CV3 - PV1 after tuning is completed.	
CV1PV2TCTuned	REAL	The calculated value of the internal model time constant for CV1 - PV2 after tuning is completed.	
CV2PV2TCTuned	REAL	The calculated value of the internal model time constant for CV2 - PV2 after tuning is completed.	
CV3PV2TCTuned	REAL	The calculated value of the internal model time constant for CV3 - PV2 after tuning is completed.	
CV1PV1DTTuned	REAL	The calculated value of the internal model deadtime for CV1 - PV1 after tuning is completed.	
CV2PV1DTTuned	REAL	The calculated value of the internal model deadtime	

Output Parameters	Data Type	Description	Values
		for CV2 - PV1 after tuning is completed.	
CV3PV1DTTuned	REAL	The calculated value of the internal model deadtime for CV3 - PV1 after tuning is completed.	
CV1PV2DTTuned	REAL	The calculated value of the internal model deadtime for CV1 - PV2 after tuning is completed.	
CV2PV2DTTuned	REAL	The calculated value of the internal model deadtime for CV2 - PV2 after tuning is completed.	
CV3PV2DTTuned	REAL	The calculated value of the internal model deadtime for CV3 - PV2 after tuning is completed.	
CV1PV1RespTCTunedS	REAL	The calculated value of the control variable time constant in slow response speed for CV1 - PV1 after tuning is completed.	
CV2PV1RespTCTunedS	REAL	The calculated value of the control variable time constant in slow response speed for CV2 - PV1 after tuning is completed.	
CV3PV1RespTCTunedS	REAL	The calculated value of the control variable time constant in slow response speed for CV3 - PV1 after tuning is completed.	
CV1PV2RespTCTunedS	REAL	The calculated value of the control variable time constant in slow response speed for CV1 - PV2 after tuning is completed.	
CV2PV2RespTCTunedS	REAL	The calculated value of the control variable time constant in slow response	

Output Parameters	Data Type	Description	Values
		speed for CV2 - PV2 after tuning is completed.	
CV3PV2RespTCTunedS	REAL	The calculated value of the control variable time constant in slow response speed for CV3 - PV2 after tuning is completed.	
CV1PV1RespTCTunedM	REAL	The calculated value of the control variable time constant in medium response speed for CV1 - PV1 after tuning is completed.	
CV2PV1RespTCTunedM	REAL	The calculated value of the control variable time constant in medium response speed for CV2 - PV1 after tuning is completed.	
CV3PV1RespTCTunedM	REAL	The calculated value of the control variable time constant in medium response speed for CV3 - PV1 after tuning is completed.	
CV1PV2RespTCTunedM	REAL	The calculated value of the control variable time constant in medium response speed for CV1 - PV2 after tuning is completed.	
CV2PV2RespTCTunedM	REAL	The calculated value of the control variable time constant in medium response speed for CV2 - PV2 after tuning is completed.	
CV3PV2RespTCTunedM	REAL	The calculated value of the control variable time constant in medium response speed for CV3 - PV2 after tuning is completed.	

Output Parameters	Data Type	Description	Values
CV1PV1RespTCTunedF	REAL	The calculated value of the control variable time constant in fast response speed for CV1 - PV1 after tuning is completed.	
CV2PV1RespTCTunedF	REAL	The calculated value of the control variable time constant in fast response speed for CV2 - PV1 after tuning is completed.	
CV3PV1RespTCTunedF	REAL	The calculated value of the control variable time constant in fast response speed for CV3 - PV1 after tuning is completed.	
CV1PV2RespTCTunedF	REAL	The calculated value of the control variable time constant in fast response speed for CV1 - PV2 after tuning is completed.	
CV2PV2RespTCTunedF	REAL	The calculated value of the control variable time constant in fast response speed for CV2 - PV2 after tuning is completed.	
CV3PV2RespTCTunedF	REAL	The calculated value of the control variable time constant in fast response speed for CV3 - PV2 after tuning is completed.	
AtuneCV1PV1On	BOOL	Set True when auto tuning for CV1 - PV1 has been initiated.	
AtuneCV2PV1On	BOOL	Set True when auto tuning for CV2 - PV1 has been initiated.	
AtuneCV3PV1On	BOOL	Set True when auto tuning for CV3 - PV1 has been initiated.	
AtuneCV1PV1Done	BOOL	Set True when auto tuning for CV1 - PV1 has completed successfully.	

Output Parameters	Data Type	Description	Values
AtuneCV2PV1Done	BOOL	Set True when auto tuning for CV2 - PV1 has completed successfully.	
AtuneCV3PV1Done	BOOL	Set True when auto tuning for CV3 - PV1 has completed successfully.	
AtuneCV1PV1Aborted	BOOL	Set True when auto tuning for CV1 - PV1 has been aborted by user or due to errors that occurred during the auto tuning operation.	
AtuneCV2PV1Aborted	BOOL	Set True when auto tuning for CV2 - PV1 has been aborted by user or due to errors that occurred during the auto tuning operation.	
AtuneCV3PV1Aborted	BOOL	Set True when auto tuning for CV3 PV1 has been aborted by user or due to errors that occurred during the auto tuning operation.	
AtuneCV1PV2On	BOOL	Set True with auto tuning for CV1 - PV2 has be initiated.	
AtuneCV2PV2On	BOOL	Set True with auto tuning for CV2 - PV2 has be initiated.	
AtuneCV3PV2On	BOOL	Set True with auto tuning for CV3 - PV2 has be initiated.	
AtuneCV1PV2Done	BOOL	Set True when auto tuning for CV1 - PV2 has completed successfully.	
AtuneCV2PV2Done	BOOL	Set True when auto tuning for CV2 - PV2 has completed successfully.	

Output Parameters	Data Type	Description	Values
AtuneCV3PV2Done	BOOL	Set True when auto tuning for CV3 - PV2 has completed successfully.	
ATuneCV1PV2Aborted	BOOL	Set True when auto tuning for CV1-PV2 has been aborted by user or due to errors that occurred during the auto tuning operation.	
ATuneCV2PV2Aborted	BOOL	Set True when auto tuning for CV2-PV2 has been aborted by user or due to errors that occurred during the auto tuning operation.	
ATuneCV3PV2Aborted	BOOL	Set True when auto tuning for CV3-PV2 has been aborted by user or due to errors that occurred during the auto tuning operation.	
AtuneCV1PV1Status	DINT	Bit mapped status for CV1 - PV1. A value of 0 indicates that no faults have occurred.	
AtuneCV2PV1Status	DINT	Bit mapped status for CV2 - PV1. A value of 0 indicates that no faults have occurred.	
AtuneCV3PV1Status	DINT	Bit mapped status for CV3 - PV1. A value of 0 indicates that no faults have occurred.	
AtuneCV1PV2Status	DINT	Bit mapped status for CV1 - PV2. A value of 0 indicates that no faults have occurred.	
AtuneCV2PV2Status	DINT	Bit mapped status for CV2 - PV2. A value of 0 indicates that no faults have occurred.	
AtuneCV3PV2Status	DINT	Bit mapped status for CV3 - PV2. A value of 0	

Output Parameters	Data Type	Description	Values
		indicates that no faults have occurred.	
AtuneCV1PV1Fault	BOOL	CV1 - PV1 Autotune has generated any of the following faults.	Bit 0 of AtuneCV1PV1Status
AtuneCV2PV1Fault	BOOL	CV2 - PV1 Autotune has generated any of the following faults.	Bit 0 of AtuneCV2PV1Status
AtuneCV3PV1Fault	BOOL	CV3 - PV1 Autotune has generated any of the following faults.	Bit 0 of AtuneCV3PV1Status
AtuneCV1PV1OutOfLimit	BOOL	Either PV1 or the deadtime-step ahead prediction of PV1 exceeds PV1TuneLimit during CV1 - PV1 Autotuning. When True, CV1 - PV1 Autotuning is aborted.	Bit 1 of AtuneCV1PV1Status
AtuneCV2PV1OutOfLimit	BOOL	Either PV1 or the deadtime-step ahead prediction of PV1 exceeds PV1TuneLimit during CV2 - PV1 Autotuning. When True, CV2 - PV1 Autotuning is aborted.	Bit 1 of AtuneCV2PV1Status
AtuneCV3PV1OutOfLimit	BOOL	Either PV1 or the deadtime-step ahead prediction of PV1 exceeds PV1TuneLimit during CV3 - PV1 Autotuning. When True, CV3 - PV1 Autotuning is aborted.	Bit 1 of AtuneCV3PV1Status
AtuneCV1PV1ModeInv	BOOL	The MMC mode was not Manual at start of Autotuning or the MMC mode was changed from Manual during CV1 - PV1 Autotuning. When True, CV1 - PV1 Autotuning is not started or is aborted.	Bit 2 of AtuneCV1PV1Status
AtuneCV2PV1ModeInv	BOOL	The MMC mode was not Manual at start of Autotuning or the MMC mode was changed from	Bit 2 of AtuneCV2PV1Status

Output Parameters	Data Type	Description	Values
		Manual during CV2 - PV1 Autotuning. When True, CV2 - PV1 Autotuning is not started or is aborted.	
AtuneCV3PV1ModeInv	BOOL	The MMC mode was not Manual at start of Autotuning or the MMC mode was changed from Manual during CV3 - PV1 Autotuning. When True, CV3 - PV1 Autotuning is not started or is aborted.	Bit 2 of AtuneCV3PV1Status
AtuneCV1PV1WindupFault	BOOL	CV1WindupHIn or CV1WindupLIn is True at start of CV1 - PV1 Autotuning or during CV1 - PV1 Autotuning. When True, CV1 - PV1 Autotuning is not started or is aborted.	Bit 3 of AtuneCV1PV1Status
AtuneCV2PV1WindupFault	BOOL	CV2WindupHIn or CV2WindupLIn is True at start of CV2 - PV1 Autotuning or during CV2 - PV1 Autotuning. When True, CV2 - PV1 Autotuning is not started or is aborted.	Bit 3 of AtuneCV2PV1Status
AtuneCV3PV1WindupFault	BOOL	CV3WindupHIn or CV3WindupLIn is True at start of CV3 - PV1 Autotuning or during CV3 - PV1 Autotuning. When True, CV3 - PV1 Autotuning is not started or is aborted.	Bit 3 of AtuneCV3PV1Status
AtuneCV1PV1StepSize0	BOOL	CV1StepSizeUsed = 0 at start of CV1 - PV1 Autotuning. When True, CV1 - PV1 Autotuning is not started.	Bit 4 of AtuneCV1PV1Status
AtuneCV2PV1StepSize0	BOOL	CV2StepSizeUsed = 0 at start of CV2 - PV1 Autotuning. When True, CV2 - PV1 Autotuning is not started.	Bit 4 of AtuneCV2PV1Status
AtuneCV3PV1StepSize0	BOOL	CV3StepSizeUsed = 0 at start of CV3 - PV1	Bit 4 of AtuneCV3PV1Status

Output Parameters	Data Type	Description	Values
		Autotuning. When True, CV3 - PV1 Autotuning is not started.	
AtuneCV1PV1LimitsFault	BOOL	CV1LimitsInv and CVManLimiting are True at start of CV1 - PV1 Autotuning or during CV1 - PV1 Autotuning. When True, CV1 - PV1 Autotuning is not started or is aborted.	Bit 5 of AtuneCV1PV1Status
AtuneCV2PV1LimitsFault	BOOL	CV2LimitsInv and CVManLimiting are True at start of CV2 - PV1 Autotuning or during CV2 - PV1 Autotuning. When True, CV2 _PV1 Autotuning is not started or is aborted.	Bit 5 of AtuneCV2PV1Status
AtuneCV3PV1LimitsFault	BOOL	CV3LimitsInv and CVManLimiting are True at start of CV3 - PV1 Autotuning or during CV3 - PV1 Autotuning. When True, CV3 - PV1 Autotuning is not started or is aborted.	Bit 5 of AtuneCV3PV1Status
AtuneCV1PV1InitFault	BOOL	CV1Initializing is True at start of CV1 - PV1 Autotuning or during CV1 - PV1 Autotuning. When True, CV1 - PV1 Autotuning is not started or is aborted.	Bit 6 of AtuneCV1PV1Status
AtuneCV2PV1InitFault	BOOL	CV2Initializing is True at start of CV2 - PV1 Autotuning or during CV2 - PV1 Autotuning. When True, CV2 - PV1 Autotuning is not started or is aborted.	Bit 6 of AtuneCV2PV1Status
AtuneCV3PV1InitFault	BOOL	CV3Initializing is True at start of CV3 - PV1 Autotuning or during CV3 - PV1 Autotuning. When True, CV3 - PV1 Autotuning is not started or is aborted.	Bit 6 of AtuneCV3PV1Status
AtuneCV1PV1EUSpanChanged	BOOL	CV1EUSpan or PV1EUSpan changes during CV1 - PV1 Autotuning. When True,	Bit 7 of AtuneCV1PV1Status

Output Parameters	Data Type	Description	Values
		CV1 - PV1 Autotuning is aborted.	
AtuneCV2PV1EUSpanChanged	BOOL	CV2EUSpan or PV1EUSpan changes during CV2 - PV1 Autotuning. When True, CV2 - PV1 Autotuning is aborted.	Bit 7 of AtuneCV2PV1Status
AtuneCV3PV1EUSpanChanged	BOOL	CV3EUSpan or PV1EUSpan changes during CV3 - PV1 Autotuning. When True, CV3 - PV1 Autotuning is aborted.	Bit 7 of AtuneCV3PV1Status
AtuneCV1PV1Changed	BOOL	CV1Oper is changed when in Operator control or CV1Prog is changed when in Program control or CV1 becomes high/low or ROC limited during CV1 - PV1 Autotuning. When True, CV1 - PV1 Autotuning is aborted.	Bit 8 of AtuneCV1PV1Status
AtuneCV2PV1Changed	BOOL	CV2Oper is changed when in Operator control or CV2Prog is changed when in Program control or CV2 becomes high/low or ROC limited during CV2 - PV1 Autotuning. When True, CV2 - PV1 Autotuning is aborted.	Bit 8 of AtuneCV2PV1Status
AtuneCV3PV1Changed	BOOL	CV3Oper is changed when in Operator control or CV3Prog is changed when in Program control or CV3 becomes high/low or ROC limited during CV3 - PV1 Autotuning. When True, CV3 - PV1 Autotuning is aborted.	Bit 8 of AtuneCV3PV1Status
AtuneCV1PV1Timeout	BOOL	Elapsed time is greater then PV1AtuneTimeLimit since step test is started. When True, CV1 - PV1 Autotuning is aborted.	Bit 9 of AtuneCV1PV1Status

Output Parameters	Data Type	Description	Values
AtuneCV2PV1Timeout	BOOL	Elapsed time is greater then PV1AtuneTimeLimit since step test is started. When True, CV2 - PV1 Autotuning is aborted.	Bit 9 of AtuneCV2PV1Status
AtuneCV3PV1Timeout	BOOL	Elapsed time is greater then PV1AtuneTimeLimit since step test is started. When True, CV3 - PV1 Autotuning is aborted.	Bit 9 of AtuneCV3PV1Status
AtuneCV1PV1NotSettled	BOOL	The PV1 is changed too much to Autotune for CV1 - PV1. When True, CV1 - PV1 Autotuning is aborted. Wait until PV1 is more stable before autotuning CV1 - PV1.	Bit 10 of AtuneCV1PV1Status
AtuneCV2PV1NotSettled	BOOL	The PV1 is changed too much to Autotune for CV2 - PV1. When True, CV2 - PV1 Autotuning is aborted. Wait until PV1 is more stable before autotuning CV2 - PV1.	Bit 10 of AtuneCV2PV1Status
AtuneCV3PV1NotSettled	BOOL	The PV1 is changed too much to Autotune for CV3 - PV1. When True, CV3 - PV1 Autotuning is aborted. Wait until PV1 is more stable before autotuning CV3 - PV1.	Bit 10 of AtuneCV3PV1Status
AtuneCV1PV2Fault	BOOL	CV1 - PV2 Autotune has generated any of the following faults.	Bit 0 of AtuneCV1PV2Status
AtuneCV2PV2Faul	BOOL	CV2 - PV2 Autotune has generated any of the following faults.	Bit 0 of AtuneCV2PV2Status
AtuneCV3PV2Fault	BOOL	CV3 - PV2 Autotune has generated any of the following faults.	Bit 0 of AtuneCV3PV2Status
AtuneCV1PV2OutOfLimit	BOOL	Either PV2 or the deadtime-step ahead prediction of PV2 exceeds PV2TuneLimit during CV1	Bit 1 of AtuneCV1PV2Status

Output Parameters	Data Type	Description	Values
		- PV2 Autotuning. When True, CV1 - PV2 Autotuning is aborted.	
AtuneCV2PV2OutOfLimit	BOOL	Either PV2 or the deadtime-step ahead prediction of PV2 exceeds PV2TuneLimit during CV2 - PV2 Autotuning. When True, CV2 - PV2 Autotuning is aborted.	Bit 1 of AtuneCV2PV2Status
AtuneCV3PV2OutOfLimit	BOOL	Either PV2 or the deadtime-step ahead prediction of PV2 exceeds PV2TuneLimit during CV3 - PV2 Autotuning. When True, CV3 - PV2 Autotuning is aborted.	Bit 1 of AtuneCV3PV2Status
AtuneCV1PV2ModeInv	BOOL	The MMC mode was not Manual at start of Autotuning or the MMC mode was changed from Manual during CV1-PV2 Autotuning. When True, CV1-PV2 Autotuning is not started or is aborted.	Bit 2 of AtuneCV1PV2Status
AtuneCV2PV2ModeInv	BOOL	The MMC mode was not Manual at start of Autotuning or the MMC mode was changed from Manual during CV2-PV2 Autotuning. When True, CV2-PV2 Autotuning is not started or is aborted.	Bit 2 of AtuneCV2PV2Status
AtuneCV3PV2ModeInv	BOOL	The MMC mode was not Manual at start of Autotuning or the MMC mode was changed from Manual during CV3-PV2 Autotuning. When True, CV3-PV2 Autotuning is not started or is aborted.	Bit 2 of AtuneCV3PV2Status
AtuneCV1PV2WindupFault	BOOL	CV1WindupHIn or CV1WindupLIn is True at start of CV1 - PV2 Autotuning or during CV1 - PV2 Autotuning. When	Bit 3 of AtuneCV1PV2Status

Output Parameters	Data Type	Description	Values
		True, CV1 - PV2 Autotuning is not started or is aborted.	
AtuneCV2PV2WindupFault	BOOL	CV2WindupHIn or CV2WindupLIn is True at start of CV2 - PV2 Autotuning or during CV2 - PV2 Autotuning. When True, CV2 - PV2 Autotuning is not started or is aborted.	Bit 3 of AtuneCV2PV2Status
AtuneCV3PV2WindupFault	BOOL	CV3WindupHIn or CV3WindupLIn is True at start of CV3 - PV2 Autotuning or during CV3 - PV2 Autotuning. When True, CV3 - PV2 Autotuning is not started or is aborted.	Bit 3 of AtuneCV3PV2Status
AtuneCV1PV2StepSize0	BOOL	CV1StepSizeUsed = 0 at start of CV1 - PV2 Autotuning. When True, CV1 - PV2 Autotuning is not started.	Bit 4 of AtuneCV1PV2Status
AtuneCV2PV2StepSize0	BOOL	CV2StepSizeUsed = 0 at start of CV2 - PV2 Autotuning. When True, CV2 - PV2 Autotuning is not started.	Bit 4 of AtuneCV2PV2Status
AtuneCV3PV2StepSize0	BOOL	CV3StepSizeUsed = 0 at start of CV3 - PV2 Autotuning. When True, CV3 - PV2 Autotuning is not started.	Bit 4 of AtuneC3PV2Status
AtuneCV1PV2LimitsFault	BOOL	CV1LimitsInv and CVManLimiting are True at start of CV1 - PV2 Autotuning or during CV1 - PV2 Autotuning. When True, CV1 - PV2 Autotuning is not started or is aborted.	Bit 5 of AtuneCV1PV2Status
AtuneCV2PV2LimitsFault	BOOL	CV2LimitsInv and CVManLimiting are True at start of CV2 - PV2 Autotuning or during CV2 - PV2 Autotuning. When	Bit 5 of AtuneCV2PV2Status

Output Parameters	Data Type	Description	Values
		True, CV2 - PV2 Autotuning is not started or is aborted.	
AtuneCV3PV2LimitsFault	BOOL	CV3LimitsInv and CVManLimiting are True at start of CV3 - PV2 Autotuning or during CV3 - PV2 Autotuning. When True, CV3 - PV2 Autotuning is not started or is aborted.	Bit 5 of AtuneCV3PV2Status
AtuneCV1PV2InitFault	BOOL	CV1Initializing is True at start of CV1 - PV2 Autotuning or during CV1 - PV2 Autotuning. When True, CV1 - PV2 Autotuning is not started or is aborted.	Bit 6 of AtuneCV1PV2Status
AtuneCV2PV2InitFault	BOOL	CV2Initializing is True at start of CV2 - PV2 Autotuning or during CV2 - PV2 Autotuning. When True, CV2 - PV2 Autotuning is not started or is aborted.	Bit 6 of AtuneCV2PV2Status
AtuneCV3PV2InitFault	BOOL	CV3Initializing is True at start of CV3 - PV2 Autotuning or during CV3 - PV2 Autotuning. When True, CV3 - PV2 Autotuning is not started or is aborted.	Bit 6 of AtuneCV3PV2Status
AtuneCV1PV2EUSpanChanged	BOOL	CV1EUSpan or PV2EUSpan changes during CV1 - PV2 Autotuning. When True, CV1 - PV2 Autotuning is aborted.	Bit 7 of AtuneCV1PV2Status
AtuneCV2PV2EUSpanChanged	BOOL	CV2EUSpan or PV2EUSpan changes during CV2 - PV2 Autotuning. When True, CV2 - PV2 Autotuning is aborted.	Bit 7 of AtuneCV2PV2Status
AtuneCV3PV2EUSpanChanged	BOOL	CV3EUSpan or PV2EUSpan changes during CV2 - PV3 Autotuning. When True, CV3 - PV2 Autotuning is aborted.	Bit 7 of AtuneCV3PV2Status
AtuneCV1PV2Changed	BOOL	CV1Oper is changed when in Operator control or	Bit 8 of AtuneCV1PV2Status

Output Parameters	Data Type	Description	Values
		CV1Prog is changed when in Program control or CV1 becomes high/low or ROC limited during CV1 - PV2 Autotuning. When True, CV1 - PV2 Autotuning is aborted.	
AtuneCV2PV2Changed	BOOL	CV2Oper is changed when in Operator control or CV2Prog is changed when in Program control or CV2 becomes high/low or ROC limited during CV2 - PV2 Autotuning. When True, CV2 - PV2 Autotuning is aborted.	Bit 8 of AtuneCV2PV2Status
AtuneCV3PV2Changed	BOOL	CV3Oper is changed when in Operator control or CV3Prog is changed when in Program control or CV3 becomes high/low or ROC limited during CV3 - PV2 Autotuning. When True, CV3 - PV2 Autotuning is aborted.	Bit 8 of AtuneCV3PV2Status
AtuneCV1PV2Timeout	BOOL	Elapsed time is greater then PV2AtuneTimeLimit since step test is started. When True, CV1 - PV2 Autotuning is aborted.	Bit 9 of AtuneCV1PV2Status
AtuneCV2PV2Timeout	BOOL	Elapsed time is greater then PV2AtuneTimeLimit since step test is started. When True, CV2 - PV2 Autotuning is aborted.	Bit 9 of AtuneCV2PV2Status
AtuneCV3PV2Timeout	BOOL	Elapsed time is greater then PV2AtuneTimeLimit since step test is started. When True, CV3 - PV2 Autotuning is aborted.	Bit 9 of AtuneCV3PV2Status
AtuneCV1PV2NotSettled	BOOL	The PV2 is changed too much to Autotune for CV1-PV2. When True, CV1-PV2 Autotuning is aborted. Wait until PV2	Bit 10 of AtuneCV1PV2Status

Output Parameters	Data Type	Description	Values
		is more stable before autotuning CV1-PV2.	
AtuneCV2PV2NotSettled	BOOL	The PV2 is changed too much to Autotune for CV2-PV2. When True, CV2-PV2 Autotuning is aborted. Wait until PV2 is more stable before autotuning CV2-PV2.	Bit 10 of AtuneCV2PV2Status
AtuneCV3PV2NotSettled	BOOL	The PV2 is changed too much to Autotune for CV3-PV2. When True, CV3-PV2 Autotuning is aborted. Wait until PV2 is more stable before autotuning CV3-PV2.	Bit 10 of AtuneCV3PV2Status
Status1	DINT	Bit mapped status of the function block. A value of 0 indicates that no faults have occurred. Any parameter that could be configured with an invalid value must have a status parameter bit to indicate its invalid status.	
Status2	DINT	Additional bit mapped status for the function block. A value of 0 indicates that no faults have occurred. Any parameter that could be configured with an invalid value must have a status parameter bit to indicate its invalid status.	
Status3CV1	DINT	Additional bit mapped CV1 status for the function block. A value of 0 indicates that no faults have occurred.	
Status3CV2	DINT	Additional bit mapped CV2 status for the function block. A value of 0 indicates that no faults have occurred.	

Output Parameters	Data Type	Description	Values
Status3CV3	DINT	Additional bit mapped CV3 status for the function block. A value of 0 indicates that no faults have occurred.	
InstructFault	BOOL	The function block has generated a fault. Indicates state of bits in Status1, Status2, and Status3CV(n), where (n) can be 1, 2, or 3.	Bit 0 of Status1
PV1Faulted	BOOL	Process variable PV1 health bad.	Bit 1 of Status1
PV2Faulted	BOOL	Process variable PV2 health bad.	Bit 2 of Status1
PV1SpanInv	BOOL	The span of PV1 inValid, PV1EUMax < PV1EUMin.	Bit 3 of Status1
PV2SpanInv	BOOL	The span of PV2 inValid, PV2EUMax < PV2EUMin.	Bit 4 of Status1
SP1ProgInv	BOOL	SP1Prog < SP1LLimit or > SP1HLimit. Limit value used for SP1.	Bit 5 of Status1
SP2ProgInv	BOOL	SP2Prog < SP2LLimit or > SP2HLimit. Limit value used for SP2.	Bit 6 of Status1
SP1OperInv	BOOL	SP1Oper < SP1LLimit or > SP1HLimit. Limit value used for SP1.	Bit 7 of Status1
SP2OperInv	BOOL	SP2Oper < SP2LLimit or > SP2HLimit. Limit value used for SP2.	Bit 8 of Status1
SP1LimitsInv	BOOL	Limits inValid: SP1LLimit < PV1EUMin, SP1HLimit > PV1EUMax, or SP1HLimit < SP1LLimit. If SP1HLimit < SP1LLimit, then limit value by using SP1LLimit.	Bit 9 of Status1
SP2LimitsInv	BOOL	Limits inValid: SP2LLimit < PV2EUMin, SP2HLimit > PV2EUMax, or SP2HLimit < SP2LLimit. If SP2HLimit <	Bit 10 of Status1

Output Parameters	Data Type	Description	Values
		SP2LLimit, then limit value by using SP2LLimit.	
SampleTimeTooSmall	BOOL	Model DeadTime / DeltaT must be less than or equal to 200.	Bit 11 of Status1
PV1FactorInv	BOOL	Entered value for Factor1 < 0.	Bit 12 of Status1
PV2FactorInv	BOOL	Entered value for Factor2 < 0.	Bit 13 of Status1
TimingModeInv	BOOL	Entered TimingMode inValid. If the current mode is not Override or Hand then set to Manual mode.	Bit 27 of Status2
RTSMissed	BOOL	Only used when in Real Time Sampling mode. Is TRUE when ABS(DeltaT – RTSTime) > 1 millisecond.	Bit 28 of Status2.
RTSTimeInv	BOOL	Entered RTSTime inValid.	Bit 29 of Status2.
RTSTimeStampInv	BOOL	RTSTimeStamp inValid. If the current mode is not Override or Hand, then set to Manual mode.	Bit 30 of Status2.
DeltaTInv	BOOL	DeltaT inValid. If the current mode is not Override or Hand then set to Manual mode.	Bit 31 of Status2.
CV1Faulted	BOOL	Control variable CV1 health bad.	Bit 0 of Status3CV1
CV2Faulted	BOOL	Control variable CV2 health bad.	Bit 0 of Status3CV2
CV3Faulted	BOOL	Control variable CV3 health bad.	Bit 0 of Status3CV3
CV1HandFBFaulted	BOOL	CV1 HandFB value health bad.	Bit 1 of Status3CV1
CV2HandFBFaulted	BOOL	CV2 HandFB value health bad.	Bit 1 of Status3CV2

Output Parameters	Data Type	Description	Values
CV3HandFBFaulted	BOOL	CV3 HandFB value health bad.	Bit 1 of Status3CV3
CV1ProgInv	BOOL	CV1Prog < 0 or > 100, or < CV1LLimit or > CV1HLimit when CVManLimiting is TRUE. Limit value used for CV1.	Bit 2 of Status3CV1
CV2ProgInv	BOOL	CV2Prog < 0 or > 100, or < CV2LLimit or > CV2HLimit when CVManLimiting is TRUE. Limit value used for CV2.	Bit 2 of Status3CV2
CV3ProgInv	BOOL	CV3Prog < 0 or > 100, or < CV3LLimit or > CV3HLimit when CVManLimiting is TRUE. Limit value used for CV3.	Bit 2 of Status3CV3
CV1OperInv	BOOL	CV1Oper < 0 or > 100, or < CV1LLimit or > CV1HLimit when CVManLimiting is TRUE. Limit value used for CV1.	Bit 3 of Status3CV1
CV2OperInv	BOOL	CV2Oper < 0 or > 100, or < CV2LLimit or > CV2HLimit when CVManLimiting is TRUE. Limit value used for CV2.	Bit 3 of Status3CV2
CV3OperInv	BOOL	CV3Oper < 0 or > 100, or < CV3LLimit or > CV3HLimit when CVManLimiting is TRUE. Limit value used for CV3.	Bit 3 of Status3CV3
CV1OverrideValueInv	BOOL	CV1OverrideValue < 0 or > 100. Limit value used for CV1.	Bit 4 of Status3CV1
CV2OverrideValueInv	BOOL	CV2OverrideValue < 0 or > 100. Limit value used for CV2.	Bit 4 of Status3CV2
CV3OverrideValueInv	BOOL	CV3OverrideValue < 0 or > 100. Limit value used for CV3.	Bit 4 of Status3CV3

Output Parameters	Data Type	Description	Values
CV1EUSpanInv	BOOL	The span of CV1EU invalid, CV1EUMax equals CV1EUMin.	Bit 5 of Status3CV1
CV2EUSpanInv	BOOL	The span of CV2EU invalid, CV2EUMax equals CV2EUMin.	Bit 5 of Status3CV2
CV3EUSpanInv	BOOL	The span of CV3EU invalid, CV3EUMax equals CV3EUMin.	Bit 5 of Status3CV3
CV1LimitsInv	BOOL	CV1LLimit < 0, CV1HLimit > 100, or CV1HLimit <= CV1LLimit. If CV1HLimit <= CV1LLimit, limit CV1 by using CV1LLimit.	Bit 6 of Status3CV1
CV2LimitsInv	BOOL	CV2LLimit < 0, CV2HLimit > 100, or CV2HLimit <= CV2LLimit. If CV2HLimit <= CV2LLimit, limit CV2 by using CV2LLimit.	Bit 6 of Status3CV2
CV3LimitsInv	BOOL	CV3LLimit < 0, CV3HLimit > 100, or CV3HLimit <= CV3LLimit. If CV3HLimit <= CV3LLimit, limit CV3 by using CV3LLimit.	Bit 6 of Status3CV3
CV1ROCLimitInv	BOOL	Entered value for CV1ROCLimit < 0, disables CV1 ROC limiting.	Bit 7 of Status3CV1
CV2ROCLimitInv	BOOL	Entered value for CV2ROCLimit < 0, disables CV2 ROC limiting.	Bit 7 of Status3CV2
CV3ROCLimitInv	BOOL	Entered value for CV3ROCLimit < 0, disables CV3 ROC limiting.	Bit 7 of Status3CV3
CV1HandFBInv	BOOL	CV1 HandFB < 0 or > 100. Limit value used for CV1.	Bit 8 of Status3CV1
CV2HandFBInv	BOOL	CV2 HandFB < 0 or > 100. Limit value used for CV2.	Bit 8 of Status3CV2
CV3HandFBInv	BOOL	CV3 HandFB < 0 or > 100. Limit value used for CV3.	Bit 8 of Status3CV3

Output Parameters	Data Type	Description	Values
CV1PV1ModelGainInv	BOOL	CV1PV1ModelGain is 1.#QNAN or -1.#IND. (Not A Number), or ± 1.$ (Infinity ∞).	Bit 9 of Status3CV1
CV2PV1ModelGainInv	BOOL	Entered value for CV2 - PV1 Model Gain is 1.#QNAN or -1.#IND. (Not A Number), or ± 1.$ (Infinity ∞	Bit 9 of Status3CV2
CV3PV1ModelGainInv	BOOL	Entered value for CV3 - PV1 Model Gain is 1.#QNAN or -1.#IND. (Not A Number), or ± 1.$ (Infinity ∞).	Bit 9 of Status3CV3
CV1PV2ModelGainInv	BOOL	CV1PV2ModelGain is 1.#QNAN or -1.#IND. (Not A Number), or ± 1.$ (Infinity ∞).	Bit 10 of Status3CV1
CV2PV2ModelGainInv	BOOL	Entered value for CV2 - PV2 Model Gain is 1.#QNAN or -1.#IND. (Not A Number), or ± 1.$ (Infinity ∞).	Bit 10 of Status3CV2
CV3PV2ModelGainInv	BOOL	Entered value for CV3 - PV2 Model Gain is 1.#QNAN or -1.#IND. (Not A Number), or ± 1.$ (Infinity ∞).	Bit 10 of Status3CV3
CV1PV1ModelTCInv	BOOL	Entered value for CV1 - PV1 Model Time Constant < 0.	Bit 11 of Status3CV1
CV2PV1ModelTCInv	BOOL	Entered value for CV2 - PV1 Model Time Constant < 0.	Bit 11 of Status3CV2
CV3PV1ModelTCInv	BOOL	Entered value for CV3 - PV1 Model Time Constant < 0.	Bit 11 of Status3CV3
CV1PV2ModelTCInv	BOOL	Entered value for CV1 - PV2 Model Time Constant < 0.	Bit 12 of Status3CV1
CV2PV2ModelTCInv	BOOL	Entered value for CV2 - PV2 Model Time Constant < 0.	Bit 12 of Status3CV2
CV3PV2ModelTCInv	BOOL	Entered value for CV3 - PV2 Model Time Constant < 0.	Bit 12 of Status3CV3

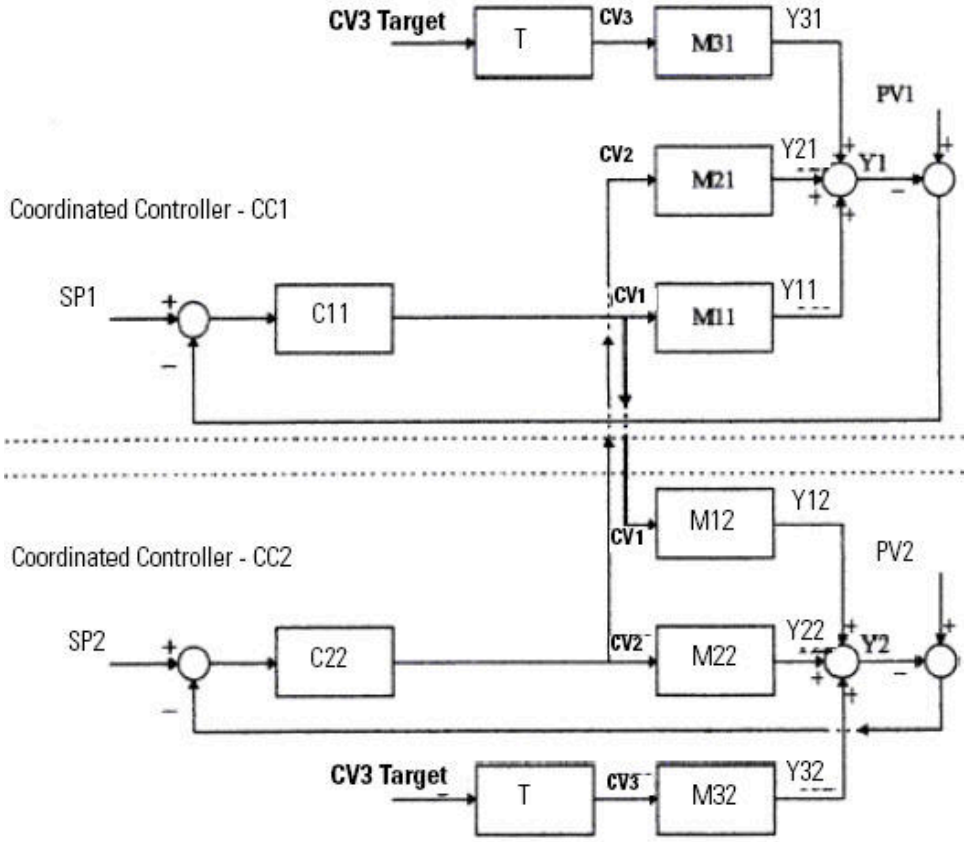
Output Parameters	Data Type	Description	Values
CV1PV1ModelDTInv	BOOL	Entered value for CV1-PV1 Model Deadtime < 0.	Bit 13 of Status3CV1
CV2PV1ModelDTInv	BOOL	Entered value for CV2-PV1 Model Deadtime < 0.	Bit 13 of Status3CV2
CV3PV1ModelDTInv	BOOL	Entered value for CV3 - PV1 Model Deadtime < 0.	Bit 13 of Status3CV3
CV1PV2ModelDTInv	BOOL	Entered value for CV1-PV2 Model Deadtime < 0.	Bit 14 of Status3CV1
CV2PV2ModelDTInv	BOOL	Entered value for CV2-PV2 Model Deadtime < 0.	Bit 14 of Status3CV2
CV3PV2ModelDTInv	BOOL	Entered value for CV3 - PV2 Model Deadtime < 0.	Bit 14 of Status3CV3
CV1PV1RespTCInv	BOOL	Entered value for CV1-PV1 Response Time Constant < 0.	Bit 15 of Status3CV1
CV2PV1RespTCInv	BOOL	Entered value for CV2-PV1 Response Time Constant < 0.	Bit 15 of Status3CV2
CV3PV1RespTCInv	BOOL	Entered value for CV3 - PV1 Response Time Constant < 0.	Bit 15 of Status3CV3
CV1PV2RespTCInv	BOOL	Entered value for CV1-PV2 Response Time Constant < 0.	Bit 16 of Status3CV1
CV2PV2RespTCInv	BOOL	Entered value for CV2-PV2 Response Time Constant < 0.	Bit 16 of Status3CV2
CV3PV2RespTCInv	BOOL	Entered value for CV3 - PV2 Response Time Constant < 0.	Bit 16 of Status3CV3
CV1TargetInv	BOOL	Entered value for CV1 Target < 0. or > 100.	Bit 17 of Status3CV1
CV2TargetInv	BOOL	Entered value for CV2 Target < 0. or > 100.	Bit 17 of Status3CV2
CV3TargetInv	BOOL	Entered value for CV3 Target < 0. or > 100.	Bit 17 of Status3CV3

Description

The MMC is a flexible model-based algorithm that can be used in two basic configuration modes:

- Three control variables used to control two interacting process variables
- Two control variables used to control two interacting process variables

Following is an MMC function block splitter example configuration.



Item	Description
M11	Internal model CV1 - PV1
M21	Internal model CV2 - PV1
M31	Internal model CV3 - PV1
M12	Internal model CV1 - PV2
M22	Internal model CV2 - PV2
M32	Internal model CV3 - PV2
T	Target response
C11, C22	Model-predictive function blocks (IMC) currently controlling PV1 and PV2 to SP1 and SP2, respectively

Y11, Y21, Y31, Y12, Y22, Y32	Model outputs of M11, M21, M31, M12, M22, M32
Y1	PV1 prediction
Y2	PV2 prediction
CV1 (Reflux ratio)	Controls PV1 (Top composition) in Coordinated Control (CC1).
CV2 (Stream Flow)	Controls PV2 (Bottom composition) in Coordinated Control (CC2)
CV3	Drives the Target value through a target response.

Affects Math Status Flags

No

Major/Minor Faults

None specific to this instruction. See Index Through Arrays on page for Array-Indexing Faults.

Execution

Note that in Structured Text, EnableIn is always true during a normal scan. Therefore, if the instruction is in the control path activated by the logic, it will execute.

Refer to Function Block Attributes for more details including definitions and general behavior for all Function Block instructions.

All conditions below the shaded area can only occur during Normal Scan mode.

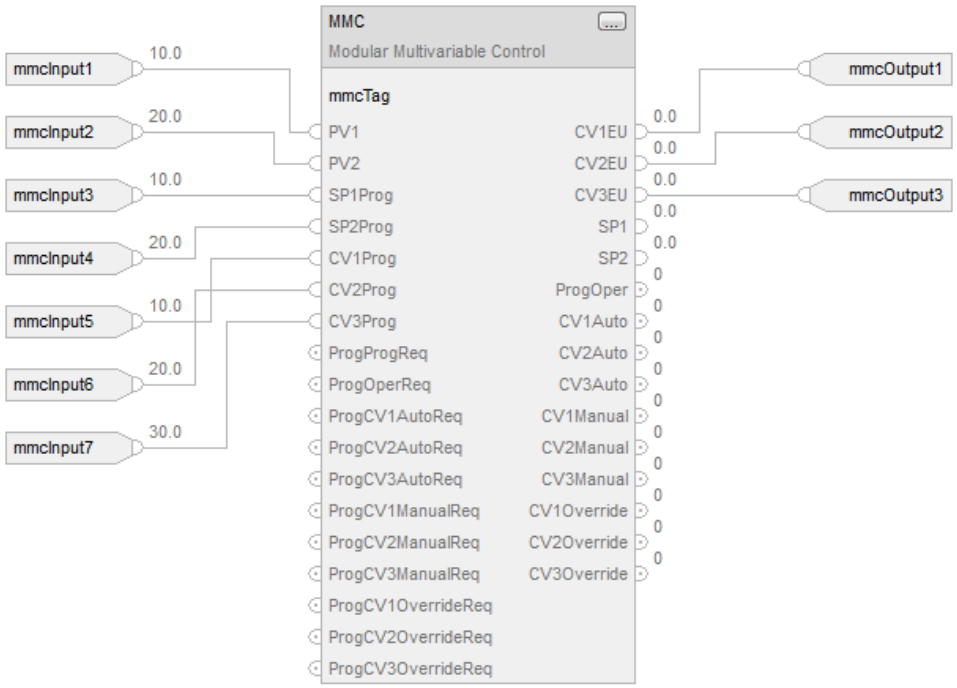
Condition/State	Action Taken
Prescan	.EnableIn and .EnableOut bits are cleared to false.
Postscan	.EnableIn and .EnableOut bits are cleared to false.
EnableIn is false	.EnableIn and .EnableOut bits are cleared to false.
Instruction first run	No state specific action taken. Primary algorithm not executed, however will validate input parameters.
Instruction first scan	No state specific action taken. Primary algorithm not executed, however will validate input parameters.
EnableIn is true	.EnableIn and .EnableOut bits are set to true. The instruction's main algorithm will be executed and outputs will be updated.

Native Implementation

Platform	Intrinsics/Main Function
ABRisc / ARM	ABRisc assembly code void FB_ModularMultivariableControl(UINT32 *pulArgOPtr)
RCA	MMC(instance) void FB_ModularMultivariableControl(UINT32 *pulArgOPtr)
SoftLogix (X86)	void rts$MMC(UINT32 *pFbdBlock)

Example

Function Block



Structured Text

```
mmcTag.PV1 := mmcInput1;
mmcTag.PV2 := mmcInput2;
mmcTag.SP1Prog := mmcInput3;
mmcTag.SP2Prog := mmcInput4;
mmcTag.CV1Prog := mmcInput5;
mmcTag.CV2Prog := mmcInput6;
mmcTag.CV3Prog := mmcInput7;
MMC(mmcTag);
mmcOutput1 := mmcTag.CV1EU;
```

```
mmcOutput2 := mmcTag.CV2EU;
mmcOutput3 := mmcTag.CV3EU;
```

MMC Function Block Configuration

Starting with the default configuration, configure the following parameters.

Parameter	Description
PV1EUMax	Maximum scaled value for PV1.
PV1EUMin	Minimum scaled value for PV1.
PV2EUMax	Maximum scaled value for PV2.
PV2EUMin	Minimum scaled value for PV2.
SP1HLimit	SP1 high limit value, scaled in PV units.
SP1LLimit	SP1 low limit value, scaled in PV units.
SP2HLimit	SP2 high limit value, scaled in PV units.
SP2LLimit	SP2 low limit value, scaled in PV units.
CV1InitValue	An initial value of the control variable CV1 output.
CV2InitValue	An initial value of the control variable CV2 output.
CV3InitValue	An initial value of the control variable CV3 output.

If you have the process models available, you can intuitively tune the MMC function block by entering the following parameters. At this point, you have completed the basic configuration. You did not configure the built-in tuner. The function block variables are ready to be put on-line in either auto or Manual mode. For tuning, the default settings will be used.

If you do not know the process models, you need to identify the models and tune the function block by using the built-in tuner (modeler) for the function block to operate correctly in the Auto mode.

Parameter	Description
ModelGains	Nonzero numbers (negative for direct acting control variable, positive for reverse acting control variable)
ModelTimeConstants	Always positive numbers
ModelDeadtimes	Always positive numbers
RespTimeConstants	Always positive numbers
Active 1st, 2nd and 3rd CV for PV1 and PV2	Specify the order in which CV's will be used to compensate for PV - SP error.

TargetCV	Specify which CV will be driven to its target value.
CVTargetValues	Specify to which values should the control variable drive the individual CV's if selected as the TargetCV.
TargetRespTC	Specify the speed of CV's to approach the target values.

For integrating process types (such as level control and position control), internal nonintegrating models are used to approximate the integrating process. The Factor parameters are used to convert the identified integrating process models to nonintegrating internal models used for CV calculation. This is necessary to provide for stable MMC execution. The MMC function block can handle any combinations of PV1 and PV2 that are integrating or nonintegrating process types.

The function block uses first order lag with deadtime internal process models and first order filters (total of up to 24 tuning parameters-6 models, 4 parameters each) to calculate the CV's. Each CV is calculated such that each process variable (PV) follows a first order lag trajectory when approaching the setpoint value.

Speed of response depends on the value of the response time constants. The smaller the response time constants, the faster the control variable response will be. The response time constants should be set such that the PV's reach the setpoints in reasonable time based on the process dynamics. The larger that the response time constants, the slower the control variable response will be, but the control variable also becomes more robust.

In the Manual mode, the control variables (CV) are set equal to the operator-entered Manual CV parameters. For the Manual to Auto mode bumpless transfer and for safe operation of the control variable, the CV rate of change limiters are implemented such that CV's cannot move from current states by more than specified CV units at each scan.

Set the CVROCPosLimit and CVROCNegLimit to limit the CV rate of change. Rate limiting is not imposed when control variable is in Manual mode unless CVManLimiting is set.

MMC Function Block Model Initialization

A model initialization occurs:

- During First Scan of the block
- When the ModelInit request parameter is set
- When DeltaT changes

You may need to manually adjust the internal model parameters or the response time constants. You can do so by changing the appropriate parameters and setting the appropriate ModelInit bit. The internal states of the function block will be initialized, and the bit will automatically reset.

For example, if you modify the model for CV2 - PV1 model, set the CV2PV1ModelInit parameter to TRUE to initialize the CV2 - PV1 internal model parameters and for the new model to take effect.

MMC Function Block Tuning

The MMC function block is equipped with an internal tuner (modeler). The purpose of the tuner is to identify the process model parameters and to use these parameters as internal model parameters (gain, time constant, and deadtime). The tuner also calculates an optimal response time constant.

Set the tuner by configuring the following parameters for each CV - PV process.

ProcessType	Integrating (level, position control) or nonintegrating (flow, pressure control)
ProcessGainSign	Set to indicate a negative process gain (increase in output causes a decrease in PV); reset to indicate a positive process gain (increase in output causes an increase in PV).
ResponseSpeed	Slow, medium, or fast, based on control objective
NoiseLevel	An estimate of noise level on PV-low, medium, or high-such that the tuner can distinguish which PV change is a random noise and which is caused by the CV step change
StepSize	A nonzero positive or negative number defining the magnitude of CV step change in either positive or negative direction, respectively
PVTuneLimit	(only for integrating process type) in PV engineering units, defines how much of PV change that is caused by CV change to tolerate before aborting the tuning test due to exceeding this limit

The tuner is started by setting the AtuneStart bit (AtuneCV1Start, for example). You can stop the tuning by setting the appropriate AtuneAbort bit.

After the tuning is completed successfully, the appropriate GainTuned, TCTuned, DTTuned, and RespTCTuned parameters are updated with the tuning results, and the AtuneStatus code is set to indicate complete.

You can copy these parameters to the ModelGain, ModelTC, ModelDT, and RespTC, respectively, by setting the AtuneUseModel bit. The MMC function block automatically initializes the internal variables and continue normal operation. It automatically resets the AtuneUseModel bit.

Use MMC Function Block for Splitter Control

The following example describes using an MMC function block to control a splitter. Refer to the MMC Function Block Splitter Example Configuration in Module Multivariable Control (MMC).

Item	Description
PV1	Top composition (more important)
PV2	Bottom composition (less important)
Active 1st CV for PV1	CV1 (reflux ratio)
Active 2nd CV for PV1	CV3 (pressure setpoint)
Active 3rd CV for PV1	CV2 (steam (flow)
Active 1st CV for PV2	CV2

Active 2nd CV for PV2	CV3
Active 3rd CV for PV2	CV1
TargetCV	CV3 (pressure should be held constant if possible)
CV3Target	60% (of pressure range)

The MMC calculates CV1, CV2, and CV3 so that the control goals are accomplished in the following order of importance:

1. Control PV1 to SP1 (PV1 is always considered more important than PV2)
2. Control PV2 to SP2
3. Control CV3 to its target value

CV1 is selected as the most active control for PV1 and CV2 as the most active for PV2. If either CV1 or CV2 saturates or is put in Manual mode, the control variable will use CV3 to maintain PV1 and PV2 at the setpoints.

MMC Function Block Tuning Errors

If an error occurs during the tuning procedure, the tuning is aborted, and an appropriate AtuneStatus bit is set. Also, a user can abort the tuning by setting the AtuneAbort parameter.

After an abort, the CV assumes its value before the step change, and the GainTuned, TCTuned, DTTuned, and RespTCTuned parameters are not updated. The AtuneStatus parameter identifies the reason for the abort.

MMC Function Block Tuning Procedure

Follow these steps to configure the tuner.

To configure the tuner

1. Put all three CV parameters into Manual mode.
2. Set the appropriate AtuneStart parameter.

   The tuner starts collecting PV and CV data for noise calculation.
3. After collecting 60 samples (60*DeltaT) period, the tuner adds StepSize to the CV.

   After successfully collecting the PV data as a result of the CV step change, the CV assumes its value before the step change and the AtuneStatus, GainTuned, TCTuned, DTTuned, and RespTCTuned parameters are updated.
4. Set the appropriate AtuneUseModel parameter to copy the tuned parameters to the model parameters

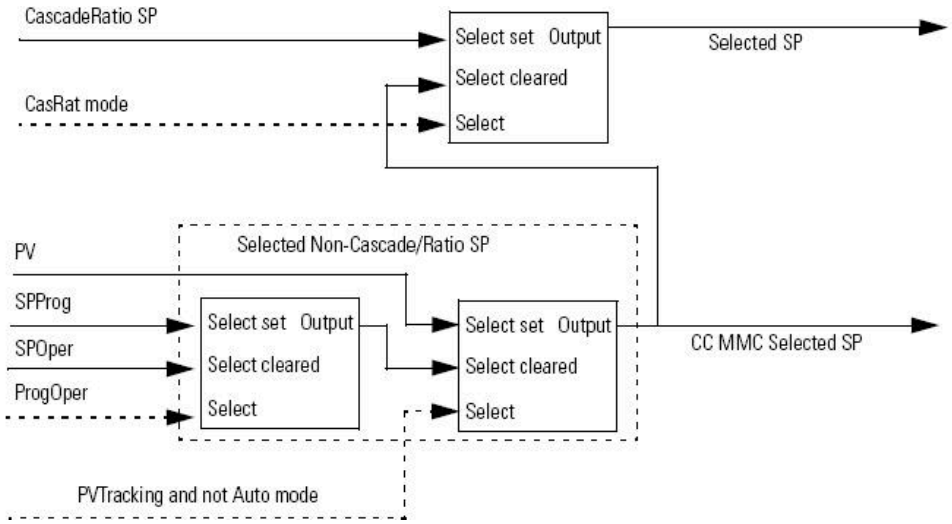
   The function block then resets the AtuneUseModel parameter.

   After a successful AutoTuneDone, the Atune parameter is set to one (1). Tuning completed successfully.

To identify models and to calculate response time constants for all six CV - PV processes, run the tuner up to three times to obtain CV1 - PV2, CV2 - PV2, and CV3 - PV2 models and tuning, respectively. After each run, two process models are identified: CV - PV1 and CV - PV2 (two process variables respond as a result of one CV step change).

Current SP

The current SP is based on the Cascade/Ratio mode, the PVTracking value, auto mode, and the ProgOper value.



Use CC Function Block to Control temperature

This is an example of how you could use the Coordinated Control function block to control the temperature in a process.

Name	Description
PV	Temperature
Act1stCV	CV3 (high pressure steam)
Act2ndCV	CV2 (cooling)
Act3rdCV	CV1 (low pressure steam)
Target1stCV	CV2
Target2ndCV	CV3
Target3rdCV	CV1
CV1Target	0% This value is irrelevant since in the target list setup, CV1 has the lowest priority, and will assume the steady state load to maintain PV at the setpoint.
CV2Target	0%
CV3Target	10%

Temperature Example Explanation

Manipulating the PV at the setpoint is the top priority. The high pressure steam and cooling are selected as the most active actuators. At steady state, the same two controls should assume their target values: CV3 at 10% and CV2 at 0%. CV1 will assume any value needed to maintain PV at the setpoint; therefore, its target value is irrelevant since manipulating the PV at the setpoint is a higher priority control objective. Target CV priorities and target CV values can be changed on-line.

The CC function block calculates CV1, CV2 and CV3 such that the control goals are accomplished in the following order of importance:

1. Control PV to SP
2. Control CV2 to its target value
3. Control CV3 to its target value

At this point, you have completed the basic configuration. You did not configure the built-in tuner. The control variable is ready to be put on-line in either auto or Manual mode. For tuning, the default settings will be used. Refer to CC Function Block Tuning.

If you do not know the process models, you need to identify the models and tune the function block by using the built-in tuner (modeler) for the function block to operate correctly in the Auto mode.

The function block uses first order lag with deadtime internal process models and first order filters (total of up to twelve tuning parameters) to calculate the CV's. Each CV is calculated such that the process variable (PV) follows a first order lag trajectory when approaching the setpoint value.

Speed of response depends on the value of the response time constants. The smaller the response time constants, the faster the control variable response will be. The response time constants should be set such that the PV reaches the setpoint in reasonable time based on the process dynamics. The larger the response time constants are, the slower the control variable response will be, but the control variable also becomes more robust. See the tuning section for more details.

In the Manual mode, the control variables (CV) are set equal to the operator-entered or program-generated CVnOper or CVnProg parameters. For the Manual to Auto mode bumpless transfer and for safe operation of the control variable, the CV rate of change limiters are implemented such that CV's cannot move from current states by more than specified CV units at each scan.

To limit the CV rate of change:

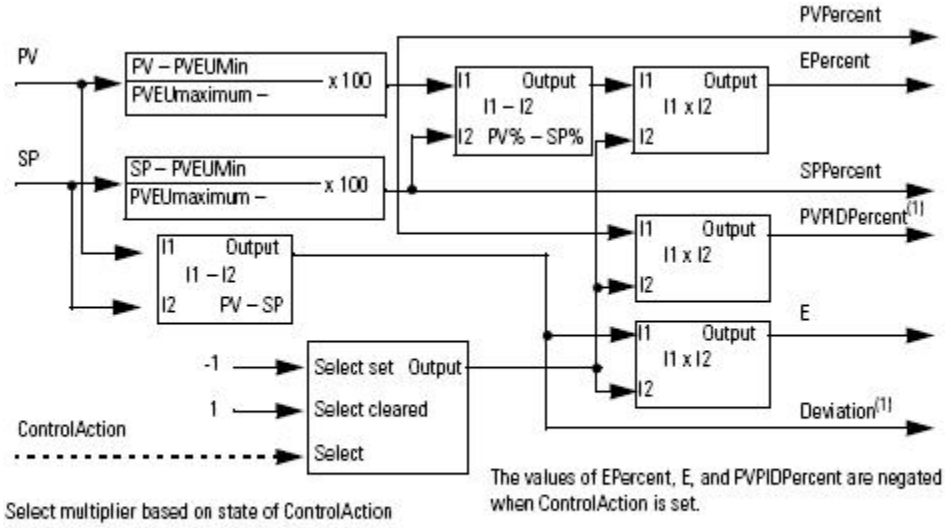
- Set the CVnROCPosLimit and CVnROCNegLimit

Rate limiting is not imposed when control variable is in Manual mode unless CVManLimiting is set.

Convert PV and SP to Percent

The instruction converts PV and SP to a percent and calculates the error before performing the PID control algorithm. The error is the difference between the PV and SP values. When

ControlAction is set, the values of EPercent, E, and PVPIDPercent are negated before being used by the PID algorithm.



(1) PVPIDPercent and Deviation are internal parameters used by the PID control algorithm.

Execution

Math status flags are set for the CV output.

Condition	Function Block Action	Structured Text Action
Prescan	InstructionFirstScan is set	InstructionFirstScan is set
Instruction First Scan	If CVFault and CVEUSpanInv are set, see Processing Faults. If CVFault and CVEUSpanInv are cleared: 1. CVInitializing is set. 2. If PVFault is set, PVSpanInv and SPLimitsInv are cleared. See Processing Faults. 3. The PID control algorithm is not executed. 4. The instruction sets CVEU = CVInitValue and CV = corresponding percentage. CVInitValue is not limited by CVEUmaximum or CVEUMin. When the instruction calculates CV as the corresponding percentage, it is limited to 0-100. $CVEU = CVInitValue$ $CV_{n-1} = CV = \frac{CVEU - CVEUMin}{CVEUMax - CVEUMin} \times 100$ $CVOper = CV$ 5. When CVInitializing and ManualAfterInit are set, the instruction disables auto and cascade/ratio modes. If the current mode is not Override or Hand mode, the instruction changes to Manual mode. If ManualAfterInit is cleared the mode is not changed.	

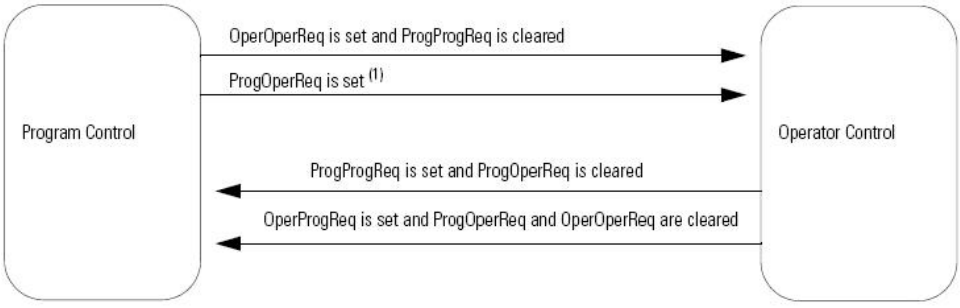
Condition	Function Block Action	Structured Text Action
	6. All the operator request inputs are cleared. 7. If ProgValueReset set, all the program request inputs are cleared 8. All the PV high-low, PV rate-of-change, and deviation high-low alarm outputs are cleared. 9. If CVInitReq is cleared, CVInitializing is cleared.	
Instruction first run	ProgOper is cleared. The instruction changes to manual mode.	ProgOper is cleared. The instruction changes to manual mode.
EnableIn is cleared	EnableOut is cleared, the instruction does nothing, and the outputs are not updated.	N/A
EnableIn is set	The instruction executes. EnableOut is set.	EnableIn is always set. The instruction executes.
Postscan	No action taken.	No action taken.

Switch Program Control to Operator

The PIDE instruction can be controlled by either a user program or an operator interface. You can change the control mode at any time. Program and

Operator control use the same ProgOper output. When ProgOper is set, control is Program; when ProgOper is cleared, control is Operator.

The following diagram shows how the PIDE instruction changes between Program control and Operator control.



(1) The instruction remains in Operator control mode when ProgOperReq is set.

For more information on program and operator control, refer to Program/Operator Control.

Operating Modes

The Cascade/Ratio, Auto, and Manual modes can be controlled by a user program when in Program control or by an operator interface when in Operator control. The Override and Hand modes have a mode request input that can only be controlled by a user program; these inputs operate in both Program and Operator control.

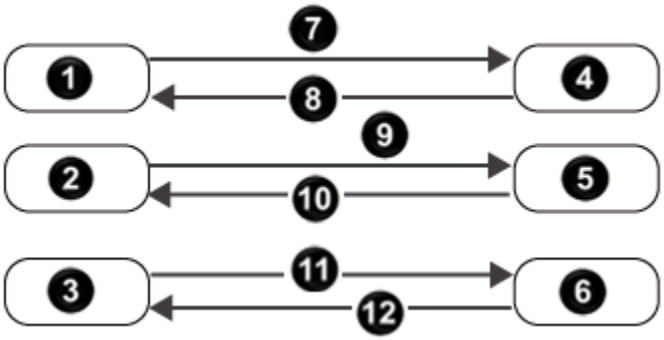
Operating Mode	Description

Cascade/Ratio	While in Cascade/Ratio mode the instruction computes the change in CV. The instruction regulates CV to maintain PV at either the SPCascade value or the SPCascade value multiplied by the Ratio value. SPCascade comes from either the CVEU of a primary PID loop for cascade control or from the "uncontrolled" flow of a ratio-controlled loop. Select Cascade/Ratio mode using either OperCasRatReq or ProgCasRatReq: Set OperCasRatReq to request Cascade/Ratio mode. Ignored when ProgOper, ProgOverrideReq, ProgHandReq, OperAutoReq, or OperManualReq is set, or when AllowCasRat is cleared. Set ProgCasRatReq to request Cascade/Ratio mode. Ignored when ProgOper or AllowCasRat is cleared or when ProgOverrideReq, ProgHandReq, ProgAutoReq, or ProgManualReq is set.
Auto	While in Auto mode the instruction computes the change in CV. The instruction regulates CV to maintain PV at the SP value. If in program control, SP = SPProg; if in Operator control, SP = SPOper. Select Auto mode using either OperAutoReq or ProgAutoReq: Set OperAutoReq to request Auto mode. Ignored when ProgOper, ProgOverrideReq, ProgHandReq, or OperManualReq is set. Set ProgAutoReq to request Auto mode. Ignored when ProgOper is cleared or when ProgOverrideReq, ProgHandReq, or ProgManualReq is set.
Manual	While in Manual mode the instruction does not compute the change in CV. The value of CV is determined by the control. If in Program control, CV = CVProg; if in Operator control, CV = CVOper. Select Manual mode using either OperManualReq or ProgManualReq: Set OperManualReq to request Manual mode. Ignored when ProgOper, ProgOverrideReq, or ProgHandReq is set. Set ProgManualReq to request Manual mode. Ignored when ProgOper is cleared or when ProgOverrideReq or ProgHandReq is set.
Override	While in Override mode the instruction does not compute the change in CV. CV = CVOverride, regardless of the control mode. Override mode is typically used to set a "safe state" for the PID loop. Select Override mode using ProgOverrideReq:

	Set ProgOverrideReq to request Override mode. Ignored when ProgHandReq is cleared.
Hand	While in Hand mode the PID algorithm does not compute the change in CV. CV = HandFB, regardless of the control mode. Hand mode is typically used to indicate that control of the final control element was taken over by a field hand/auto station. Select Hand mode using ProgHandReq: Set ProgHandReq to request hand mode. This value is usually read as a digital input from a hand/auto station.

Primary Loop Control

Primary loop control is typically used by a primary PID loop to obtain bumpless switching and anti-reset windup when using Cascade/Ratio mode. The primary loop control includes the initialize primary loop output and the anti-reset windup outputs. The InitPrimary output is typically used by the CVInitReq input of a primary PID loop. The windup outputs are typically used by the windup inputs of a primary loop to limit the windup of its CV output.



Item	Description
1	InitPrimary is cleared
2	WindupHOut is cleared(4)
3	WindupLOut is cleared(4)
4	InitPrimary is set(1)
5	WindupHOut is set
6	WindupLOut is set

Item	Description
7	CVInitializing is set or not Cascade/Ratio mode(2)
8	CVInitializing is cleared and Cascade/Ration mode(3)
9	SHPAlarm is set or appropriate Cv alarm(5)
10	SHPAlarm is cleared and no CV alarm(6)
11	SPAlarm is set or appropriate CV alarm(7)
12	SPAlarm is cleared and no CV alarm(8)

1. During first scan, the instruction sets InitPrimary.
2. When CVInitializing is set or when not in Cascade/Ratio mode, the instruction sets InitPrimary.
3. When CVInitializing is cleared and in Cascasde/Ratio mode, the instruction clears InitPrimary.
4. During instruction first scan, the instruction clears the windup outputs. The instruction also clears the windup outputs and disables the CV windup algorithm when CVInitializing is set or if either CVFaulted or CVEUSpanInv is set.
5. The instruction sets WindupHOut when SPHalarm is set, or when ControlAction is cleared and CVHAlarm is set, or when ControlAction is set and CVLAarm is set.
   The SP and CV limits operate independently. A SP high limit does not prevent CV from increasing in value. Likewise, a CV high or low limit does not prevent SP from increasing in value.
6. The instruction clears WindupHOut when SPHalarm is cleared, and not (ControlAction is cleared and CVHAlarm is set), and not (ControlAction is set and CVLAlarm is set).
7. The instruction sets WinindupLOut when SPLAlarm is set, or when ControlAction is cleared and CVLAlarm is set or when ControlAction is set and CVHAlarm is set.
   The SP and CV limits operate independently. A SP low limit does not prevent CV from increasing in value likewise a CV low or high limit does not prevent SP from increasing in value.
8. The instruction clears WindupLOut when SPLAlarm is cleared and not (ControlAction is cleared and CVLAlarm is set) and not (ControlAction is set and CVHAlarm is set).

Processing Faults

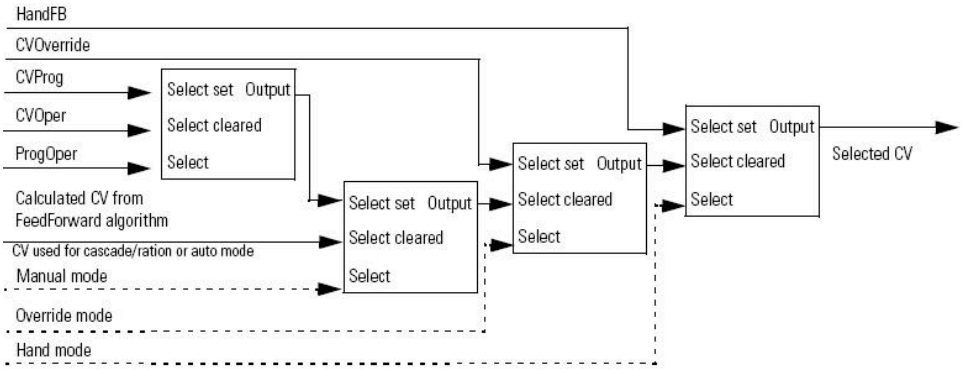
The following table describes how the instruction handles execution faults:

Fault Condition	Action
CVFaulted or CVEUSpanInv is set	• Instruction is not initialized, CVInitializing is cleared. • Compute PV and SP percent, calculate error, update internal parameters for EPercent and PVPIDPercent • PID control algorithm is not executed. • Disable the Auto and Cascade/Ratio modes. If Override or Hand is not the current mode, set to Manual mode. • Set CV to value determined by Program or Operator control and mode (Manual, Override, or Hand).
CVinitRequest	Refer to Execution.
PV Health Bad	• Disable the Auto and CasRat modes. If Override or Hand is not the current mode then set to the Manual mode. • Set PV high-low, PV rate-of-change, and deviation high-low alarm outputs FALSE • PID control algorithm is not executed. • Set CV to value by determined by Program or Operator control and mode (Manual, Override or Hand).
PVFaulted is set	• Disable the Auto and Cascade/Ratio modes. If Override or Hand is not the current mode, set to Manual mode. • PV high-low, PV rate-of-change, and deviation high-low alarm outputs are cleared • PID control algorithm is not executed. • Set CV to value by determined by Program or Operator control and mode (Manual, Override, or Hand).
PVSpanInv is set or SPLimitsInv is set	• Disable the Auto and Cascade/Ratio modes. If Override or Hand is not the current mode, set to Manual mode. • Do not compute PV and SP percent.

Fault Condition	Action
	• PID control algorithm is not executed. • Set CV to value by determined by Program or Operator control and mode (Manual, Override, or Hand).
RatioLimitsInv is set and CasRat is set and UseRatio is set	• If not already in Hand or Override, set to Manual mode. • Disable the Cascade/Ratio mode. • Set CV to value determined by Program or Operator control and mode (Manual, Override, or Hand).
TimingModeInv is set or RTSTimeStampInv is set or DeltaTInv is set	• If not already in Hand or Override, set to Manual mode.

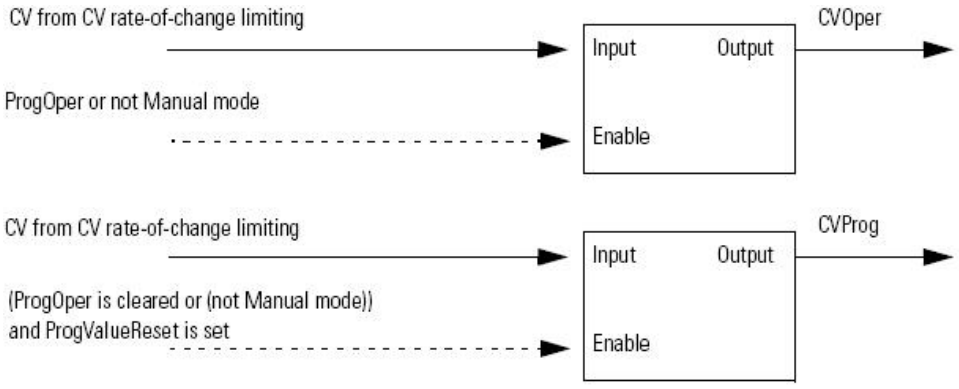
Select the Control Variable

Once the PID algorithm has been executed, select the CV based on program or operator control and the current PID mode.



Update the CVOper and CVProg Values

If not in the Operator Manual mode, the PIDE instruction sets CVOper = CV. This obtains bumpless mode switching from any control to the Operator Manual mode.



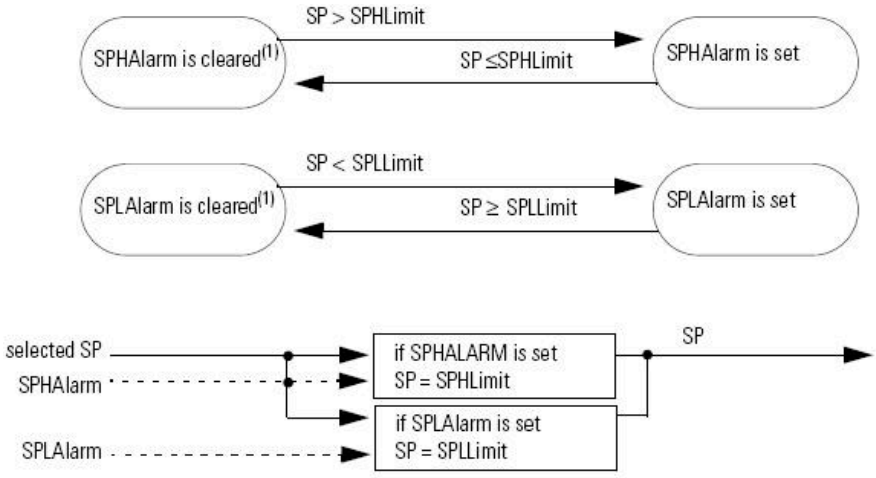
Select the Setpoint

Once the instruction determines program or operator control and the PID mode, the instruction can obtain the proper SP value.

- Enhanced PID on page 66
- Current SP on page 309
- SP High/Low Limiting on page 318

SP High/Low Limiting

The high-to-low alarming algorithm compares SP to the SPHLimit and SPLLimit alarm limits. SPHLimit cannot be greater than PVEUmaximum and SPLLimit cannot be less than PVEUMin.



(1) During instruction first scan, the instruction clears the SP alarm outputs. The instruction also clears the SP alarm limits and disables the alarming algorithm when PVSpanInv is set.

Drives Instructions

The Drives instructions include the following information.

Available Instructions

Ladder Diagram

HMIBC on page 370

Function Block and Structured Text

INTG on page 319	PI on page 326	PMUL on page 339	SCRV on page 346	SOC on page 355	UPDN on page 366	HMIBC on page 370

If you want to	Use this instruction
Execute an integral operation.	INTG
Execute a PI algorithm.	PI
Provide an interface from a position input module, such as a resolver or encoder feedback module, to the digital system by computing the change in input from one scan to the next.	PMUL
Perform a ramp function with an added jerk rate	SCRV
Use a gain term, a first order lag, and a second order lead.	SOC
Add and subtract two inputs into an accumulated value.	UPDN
Enable operators to initiate machine control operations, such as jogging a motor or enabling a valve, with a high degree of accuracy and determinism.	HMIBC

Integrator (INTG)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, CompactLogix 5380, ControlLogix 5580, and ControlLogix 5590 controllers.

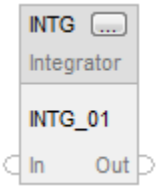
The INTG instruction implements an integral operation. This instruction is designed to execute in a task where the scan rate remains constant.

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram logic.

Function Block



Structured Text

INTG(INTG_tag);

Operands

Function Block

Operand	Type	Format	Description
INTG tag	INTEGRATOR	Structure	INTG structure

INTEGRATOR Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
In	REAL	The analog signal input to the instruction. Valid = any float Default = 0.0
Initialize	BOOL	Request to initialize control algorithm. Output = InitialValue as long as Initialize is set. Valid = any float Default = 0.0
InitialValue	REAL	The initial value for instruction. Output = InitialValue as long as Initialize is set. Valid = any float Default = 0.0
IGain	REAL	The integral gain multiplier. If IGain < 0; the instruction sets IGain = 0.0,

Input Parameter	Data Type	Description
		sets the appropriate bit in Status, and leaves the Output unchanged. Valid = 0.0 to maximum positive float Default = 0.0
HighLimit	REAL	The high limit value for Out. If HighLimit ≤ LowLimit, the instruction sets HighAlarm and LowAlarm, sets the appropriate bit in Status, and sets Out = LowLimit. Valid = any float Default = maximum positive float
LowLimit	REAL	The low limit value for Out. If HighLimit ≤ LowLimit, the instruction sets HighAlarm and LowAlarm, sets the appropriate bit in Status, and sets Out = LowLimit. Valid = any float Default = maximum negative float
HoldHigh	BOOL	Hold output high request. When set, Out is not allowed to increase in value. Default is cleared.
HoldLow	BOOL	Hold output low request. When set, Out is not allowed to decrease in value. Default is cleared.
TimingMode	DINT	Selects timing execution mode. 0 = Periodic mode 1 = Oversample mode 2 = Real time sampling mode For more information about timing modes, see Function Block Attributes. Valid = 0 to 2 Default = 0
OversampleDT	REAL	Execution time for oversample mode. Valid = 0 to 4194.303 seconds Default = 0
RTSTime	DINT	Module update period for real time sampling mode Valid = 1 to 32,767ms

Input Parameter	Data Type	Description
		Default = 1
RTSTimeStamp	DINT	Module time stamp value for real time sampling mode. Valid = 0 to 32,767ms Default = 0

Output Parameter	Data Type	Description
EnableOut	BOOL	Indicates if instruction is enabled. Cleared to false if Out overflows.
Out	REAL	The calculated output of the algorithm.
HighAlarm	BOOL	High limit alarm indicator. When Out ≥ HighLimit, HighAlarm is set and the output is limited to the value of HighLimit.
LowAlarm	BOOL	Low limit alarm indicator. When Out ≤ LowLimit, LowAlarm is set and the output is limited to the value of LowLimit.
DeltaT	REAL	Elapsed time between updates. This is the elapsed time in seconds used by the control algorithm to calculate the process output.
Status	DINT	Status of the function block.
InstructFault (Status.0)	BOOL	The instruction detected one of the following execution errors. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.
IGainInv (Status.1)	BOOL	IGain > maximum or IGain < minimum.
HighLowLimsInv (Status.2)	BOOL	HighLimit ≤ LowLimit.
TimingModeInv (Status.27)	BOOL	Invalid TimingMode value. For more information about timing modes, see Function Block Attributes.

Output Parameter	Data Type	Description
RTSMissed (Status.28)	BOOL	Only used in real time sampling mode. Set when ABS | DeltaT - RTSTime | > 1 (.001 second).
RTSTimeInv (Status.29)	BOOL	Invalid RTSTime value.
RTSTimeStampInv (Status.30)	BOOL	Invalid RTSTimeStamp value.
DeltaTInv (Status.31)	BOOL	Invalid DeltaT value.

Structured Text

Operand	Type	Format	Description
INTG tag	INTEGRATOR	Structure	INTG structure

See Structured Text Syntax for more information on the syntax of expressions within structured text.

Description

The INTG instruction is designed to execute in a task where the scan rate remains constant.

The INTG instruction executes this control algorithm when Initialize is cleared and DeltaT > 0.

$$Out = IGain \times \frac{In + In_{n-1}}{2} \times DeltaT + Out_{n-1}$$

Whenever the value computed for the output is invalid, NAN, or ± INF, the instruction sets Out = the invalid value. The internal parameters are not updated. In each subsequent scan, the output is computed using the internal parameters from the last scan when the output was valid.

Limiting

The INTG instruction performs windup limiting to stop Out from changing based on the state of the HoldHigh and HoldLow inputs. If HoldHigh is set and Out > Out_{n-1}, then Out = Out_{n-1}. If HoldLow is set and Out < Out_{n-1}, then Out = Out_{n-1}.

The INTG instruction also performs output limiting using HighLimit and LowLimit. If Out ≥ HighLimit, the Out = HighLimit and HighAlarm is set. If Out ≤ LowLimit, then Out = LowLimit and LowAlarm is set.

Affects Math Status Flags

No

Major/Minor Faults

Table 6.

Minor fault occurs when	Fault type	Fault code
Feature is enabled and overflow is detected.	4	4

See Common Attributes on page for operand-related faults.

Execution

NOTE: In Structured Text, EnableIn is always true during normal scan. Therefore, if the instruction is in the control path activated by the logic, it will execute. For more details including definitions and general behavior for all Function Block instructions, refer to Publication 1756-RM006G-EN-P, Advanced Process Control and Drives Instructions.

Condition/State	Action Taken
Prescan	No action taken.
Postscan	No action taken.

Conditions that occur only during Normal Scan mode

Condition/State	Action Taken
Instruction first run	The internal parameters and Out are set to 0. The primary algorithm is not executed but will validate input parameters.
Instruction first scan	The internal parameters and Out are set to 0. The primary algorithm is not executed but will validate input parameters.
EnableIn is false	.EnableOut bit is cleared to false.
EnableIn is true	.EnableOut bit is set to true. The instruction's main algorithm will be executed and outputs will be updated.

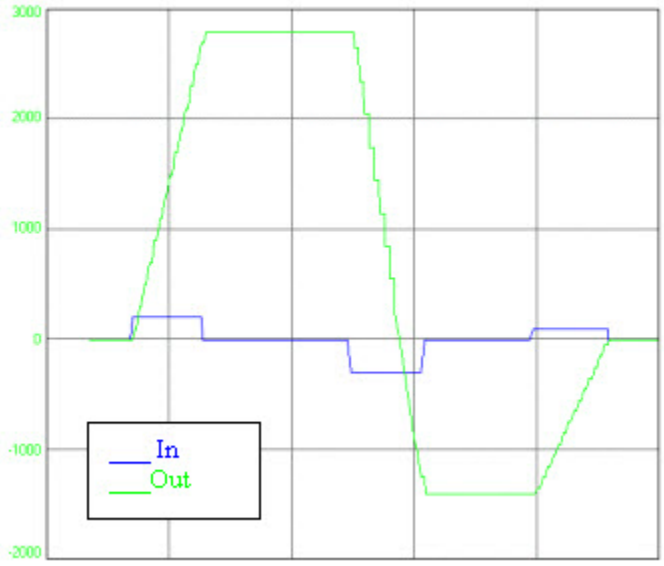
Examples

In many applications an integral gain component is included in the closed loop regulator design in order to eliminate or minimize error in the system being regulated. A straight proportional-only regulator will not tend to drive error in the system to zero. A regulator that uses proportional and integral gain, however, tends to drive the error signal to zero over a period of time. The INTG instruction uses the following equation to calculate its output.

$$Out = IGain \times \frac{In + In_{n-1}}{2} \times DeltaT + Out_{n-1}$$

In the chart below, the input to the block moves from 0 to +200 units. During this period, the output of the block integrates to 2800 units. As In changes from +200 units to 0 units, Out maintains at 2800 units. When In transitions from 0 to -300 units, Out slowly integrates down to -1400 units until In transitions back to 0. Finally, as In moves from 0 to +100, Out integrates back to 0 where In is set to 0 coincidentally with Out reaching 0.

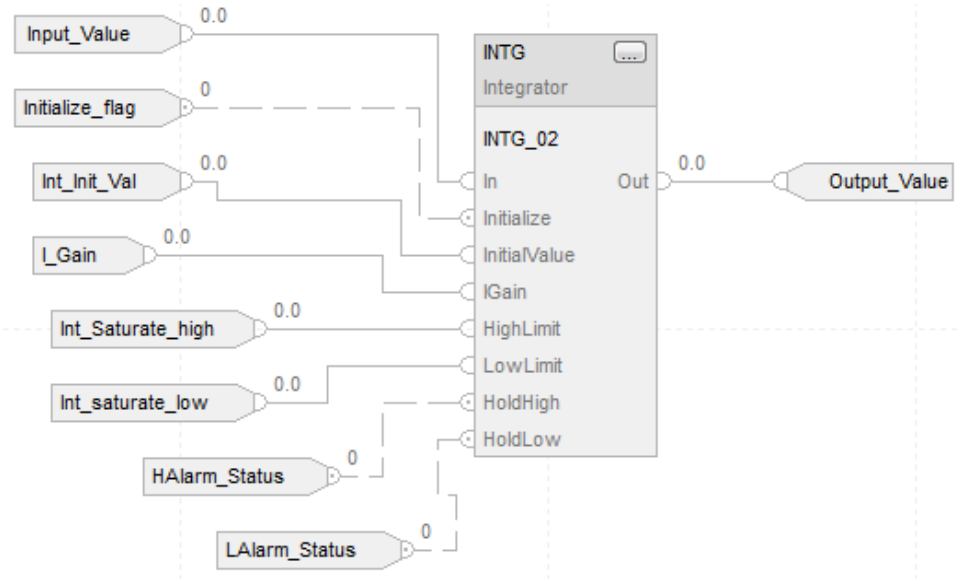
This characteristic of the integrator - continually driving in a specific direction while any input to the function is present or holding at any level during the point where the input is at zero - is what causes a regulator using integral gain to drive toward zero error over a period of time.



The following example shows how the INTG instruction can be used. In many instances, the HighLimit and LowLimit inputs limit the total percentage of control that the integral gain element might have as a function of the regulator's total output. The HoldHigh and HoldLow inputs, on the other hand, can be used to prevent the output from moving further in either the positive or negative direction. The HoldHigh and HoldLow inputs prevent the INTG instruction from "winding-up" in a direction which is already beyond the limits of the controlled variable.

Function Block

This example is the minimal legal programming of the INTG function block and is only used to show the neutral text and generated code for this instruction. This is for internal purposes only and is not a testable case.



Structured Text

```
INTG_01.In := Input_Value;
INTG_01.Initialize := Initialize_flag;
INTG_01.InitialValue := Int_Init_Val;
INTG_01.IGain := I_Gain;
INTG_01.HighLimit := Int_saturate_high;
INTG_01.LowLimit := Int_saturate_low;
INTG_01.HoldHigh :=HAlarm_Status;
INTG_01.HoldLow := LAlarm_Status;
INTG(INTG_01);
```

Proportional + Integral (PI)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, CompactLogix 5380, ControlLogix 5580, and ControlLogix 5590 controllers.

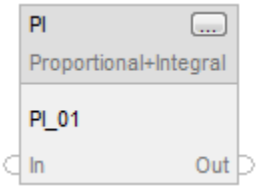
The PI instruction provides two methods of operation. The first method follows the conventional PI algorithm in that the proportional and integral gains remain constant over the range of the input signal (error). The second method uses a non-linear algorithm where the proportional and integral gains vary over the range of the input signal. The input signal is the deviation between the setpoint and feedback of the process.

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram.

Function Block



Structured Text

PI(PI_tag);

Operands

Function Block

Operand	Type	Format	Description
PI tag	PROP_INT	structure	PI structure

Structured Text

Operand	Type	Format	Description
PI tag	PROP_INT	structure	PI structure

See *Structured Text Syntax* for more information on the syntax of expressions within structured text.

PROP_INT Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
In	REAL	The process error signal input. This is the difference between setpoint and feedback. Valid = any float Default = 0.0
Initialize	BOOL	The instruction initialization command. When set, Out and

Input Parameter	Data Type	Description
		internal integrator are set equal to the value of InitialValue. Default is cleared.
InitialValue	REAL	The initial value input. When Initialize is set, Out and integrator are set to the value of InitialValue. The value of InitialValue is limited using HighLimit and LowLimit. Valid = any float Default = 0
Kp	REAL	The proportional gain. This affects the calculated value for both the proportional and integral control algorithms. If invalid, the instruction clamps Kp at the limits and sets the appropriate bit in Status. Valid = any float > 0.0 Default = minimum positive float
WId	REAL	The lead frequency in radians/second. This affects the calculated value of the integral control algorithm. If invalid, the instruction clamps WId at the limits and sets the appropriate bit in Status. Valid = see the Description section below for valid ranges Default = 0.0
HighLimit	REAL	The high limit value. This is the maximum value for Out. If HighLimit ≤ LowLimit, the instruction sets HighAlarm and LowAlarm, sets the appropriate bit in Status, and sets Out = LowLimit. Valid = LowLimit < HighLimit ≤ maximum positive float Default = maximum positive float
LowLimit	REAL	The low limit value. This is the minimum value for Out. If HighLimit ≤ LowLimit, the instruction sets HighAlarm and LowAlarm, sets the appropriate bit in Status, and sets Out = LowLimit.

Input Parameter	Data Type	Description
		Valid = maximum negative float ≤ LowLimit < HighLimit Default = maximum negative float
HoldHigh	BOOL	The hold high command. When set, the value of the internal integrator is not allowed to increase in value. Default is cleared.
HoldLow	BOOL	The hold low command. When set, the value of the internal integrator is not allowed to decrease in value. Default is cleared.
ShapeKpPlus	REAL	The positive Kp shaping gain multiplier. Used when In is ≥0. If invalid, the instruction clamps ShapeKpPlus at the limits and sets the appropriate bit in Status. Not used when NonLinearMode is cleared. Valid = 0.1 to 10.0 Default = 1.0
ShapeKpMinus	REAL	The negative Kp shaping gain multiplier. Used when In is < 0. If invalid, the instruction clamps ShapeKpMinus at the limits and sets the appropriate bit in Status. Not used when NonLinearMode is cleared. Valid = 0.1 to 10.0 Default = 1.0
KpInRange	REAL	The proportional gain shaping range. Defines the range of In (error) over which the proportional gain increases or decreases as a function of the ratio of | In | / KpInRange. When | In | > KpInRange, the instruction calculates the change in proportional error using entered the Kp shaping gain x (In - KpInRange). If invalid, the instruction clamps KpInRange at the limits and sets the appropriate bit in Status. Not used when NonLinearMode is cleared. Valid = any float > 0.0 Default = maximum positive float

Input Parameter	Data Type	Description
ShapeWIdPlus	REAL	The positive WId shaping gain multiplier. Used when In is ≥ 0. If invalid, the instruction clamps ShapeWIdPlus at the limits and sets the appropriate bit in Status. Not used when NonLinearMode is cleared. Valid = 0.0 to 10.0 Default = 1.0
ShapeWIdMinus	REAL	The negative WId shaping gain multiplier. Used when In is < 0. If invalid, the instruction clamps ShapeWIdMinus at the limits and sets the appropriate bit in Status. Not used when NonLinearMode is cleared. Valid = 0.0 to 10.0 Default = 1.0
WIdInRange	REAL	The integral gain shaping range. Defines the range of In (error) over which integral gain increases or decreases as a function of the ratio of | In | / WIdInRange. When |In| > WIdInRange, the instruction limits In to WIdInRange when calculating integral error. If invalid, the instruction clamps WIdInRange at the limits and sets the appropriate bit in Status. Not used when NonLinearMode is cleared. Valid = any float > 0.0 Default = maximum positive float
NonLinearMode	BOOL	Enable the non-linear gain mode. When set, the instruction uses the non-linear gain mode selected by ParabolicLinear to compute the actual proportional and integral gains. When cleared, the instruction disables the non-linear gain mode and uses the Kp and WId values as the proportional and integral gains. Default is cleared.
ParabolicLinear	BOOL	Selects the non-linear gain mode. The modes are linear or parabolic. When set, the instruction uses the parabolic gain method of $y=a * x^2 +$

Input Parameter	Data Type	Description
		b to calculate the actual proportional and integral gains. If cleared, the instruction uses the linear gain method of y=a * x + b. Default is cleared.
TimingMode	DINT	Selects timing execution mode. 0 = Periodic mode 1 = Oversample mode 2 = Real time sampling mode For more information about timing modes, see Function Block Attributes. Valid = 0 to 2 Default = 0
OversampleDT	REAL	Execution time for oversample mode. Valid = 0 to 4194.303 seconds Default = 0
RTSTime	DINT	Module update period for real time sampling mode Valid = 1 to 32,767ms Default = 1
RTSTimeStamp	DINT	Module time stamp value for real time sampling mode. Valid = 0 to 32,767ms Default = 0

Output Parameter	Data Type	Description
EnableOut	BOOL	Indicates the execution status of the instruction.
Out	REAL	The calculated output of the PI algorithm. Math status flags are set for this output.
HighAlarm	BOOL	The maximum limit alarm indicator. Set when the calculated value for Out ≥ HighLimit and the output and integrator are clamped at HighLimit.
LowAlarm	BOOL	The minimum limit alarm indicator. Set when the calculated value for

Output Parameter	Data Type	Description
		Out ≤ LowLimit and output and integrator are clamped at LowLimit.
DeltaT	REAL	Elapsed time between updates. This is the elapsed time in seconds used by the control algorithm to calculate the process output.
Status	DINT	Status of the function block.
InstructFault (Status.0)	BOOL	The instruction detected one of the following execution errors. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.
KpInv (Status.1)	BOOL	Kp < minimum or Kp > maximum.
WldInv (Status.2)	BOOL	Wld < minimum or Wld > maximum.
HighLowLimsInv (Status.3)	BOOL	HighLimit ≤ LowLimit.
ShapeKpPlusInv (Status.4)	BOOL	ShapeKpPlus < minimum or ShapeKpPlus > maximum.
ShapeKpMinusInv (Status.5)	BOOL	ShapeKpMinus < minimum or ShapeKpMinus > maximum.
KpInRangeInv (Status.6)	BOOL	KpInRange < minimum or KpInRange > maximum.
ShapeWldPlusInv (Status.7)	BOOL	ShapeWldPlus < minimum or ShapeWldPlus > maximum.
ShapeWldMinusInv (Status.8)	BOOL	ShapeWldMinus < minimum or ShapeWldMinus > maximum.
WldInRangeInv (Status.9)	BOOL	WldInRange < minimum or WldInRange > maximum.
TimingModeInv (Status.27)	BOOL	Invalid TimingMode. For more information about timing modes, see Function Block Attributes.
RTSMissed (Status.28)	BOOL	Only used in real time sampling mode. Set when ABS | DeltaT - RTSTime | > 1 (.001 second).
RTSTimeInv (Status.29)	BOOL	Invalid RTSTime value.

Output Parameter	Data Type	Description
RTSTimeStampInv (Status.30)	BOOL	Invalid RTSTimeStamp value.
DeltaT (Status.31)	BOOL	Invalid DeltaT value.

Description

The PI instruction uses the position form of the PI algorithm. This means the gain terms are applied directly to the input signal, rather than to the change in the input signal. The PI instruction is designed to execute in a task where the scan rate remains constant.

In the non-linear algorithm, the proportional and integral gains vary as the magnitude of the input signal changes. The PI instruction supports two non-linear gain modes: linear and parabolic. In the linear algorithm, the gains vary linearly as the magnitude of input changes. In the parabolic algorithm, the gains vary according to a parabolic curve as the magnitude of input changes.

The PI instruction calculates Out using this equation:

$$Kp \times \frac{s + Wld}{s}$$

Whenever the value computed for the output is invalid, NAN, or ± INF, the instruction sets Out = the invalid value and sets the math overflow status flag. The internal parameters are not updated. In each subsequent scan, the output is computed using the internal parameters from the last scan when the output was valid.

Operating in Linear Mode

In linear mode, the non-linear gain mode is disabled. The Kp and Wld values are the proportional and integral gains used by the instruction. The instruction calculates the value for Out using these equations:

Value	Equation
ITerm	$Kp \times Wld \times \frac{WldInput + WldInput_{n-1}}{2} \times DeltaT + ITerm_{n-1}$ where DeltaT is in seconds
PTerm	*Kp x In*
Out	*ITerm + PTerm*

with these limits on Wld:

- Low Limit > 0.0
- High Limit = 0.7p/DeltaT
- WldInput = In

Operating in Non-Linear Mode

In non-linear mode, the instruction uses the non-linear gain mode selected by ParabolicLinear to compute the actual proportional and integral gains.

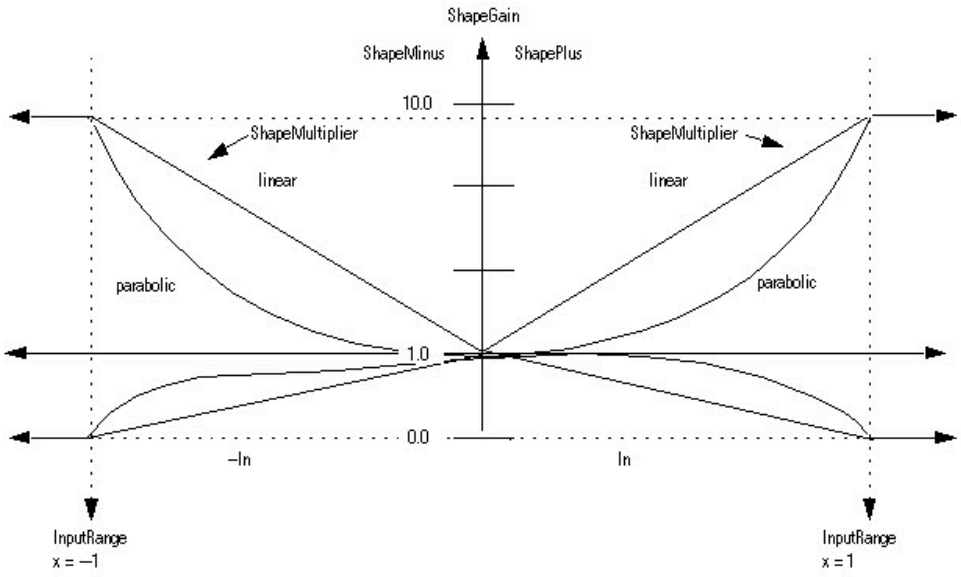
The gains specified by Kp and Wld are multiplied by 1.0 when In = 0. Separate proportional and integral algorithms increase or decrease the proportional or integral gain as the magnitude of error changes. These algorithms use the input range and shaping gain parameters to compute the actual proportional and integral gains. Input range defines the range of In (i.e. error) over which the gain is shaped. Input ranges are set by the two KpInRange and WldInRange. Shaping gain defines the gain multiplier for the quadrant controlled by the shaping gain parameter. Shaping gains are set by ShapeKpPlus, ShapeKpMinus, ShapeWldPlus and ShapeWldMinus.

The ParabolicLinear input selects the non-linear gain mode. If ParabolicLinear is cleared, linear mode is selected. If ParabolicLinear is set, parabolic mode is selected.

To configure a particular shaping gain curve, enter a shaping gain 0.0-10.0 for integral shaping, a shaping gain 0.1-10.0 for proportional shaping, and the input range over which shaping is to be applied. Kp and Wld are multiplied by the calculated ShapeMultiplier to obtain the actual proportional and integral gains. Entering a shaping gain of 1.0 disables the non-linear algorithm that calculates the proportional or integral gain for the quadrant.

When the magnitude of In (error) is greater then InRange then the ShapeMultiplier equals the value computed when | In | was equal to InRange.

The following diagram illustrates the maximum and minimum gain curves that represent the parabolic and linear gain equations.



The instruction calculates the value for Out using these equations:

Value	Equation
Kp shaping gain multiplier	If In ≥ 0 then: $KpShapeGain = ShapeKpPlus$ $KpRange = KpInRange$ Else: $KpShapeGain = ShapeKpMinus$ $KpRange = -KpInRange$
Kp input ratio	If |In| ≤ KpInRange: $KpInputRatio = \|In\| \times \frac{1}{KpInRange}$ Else: $KpInputRatio = 1$
Kp ratio	If not parabolic mode: $KpRatio = KpInputRatio \times 0.5$ If parabolic mode: $KpRatio = KpInputRatio^2 \times 0.333$
Kps shaping gain	$Kps = Kp \times (((KpShapeGain - 1) \times KpRatio) + 1)$
Proportional output	If |In| ≤ KpInRange: $PTerm = Kps \times In$ Else, limit gain: $PTerm = Kps \times KpRange + (In - KpRange) \times KpShape$
Wld shaping gain	If In ≥ 0 then: $WldShapeGain = ShapeWldPlus$ Else: $WldShapeGain = ShapeWldMinus$
Wld input	If In > WldRange then: $WldInput = WldInRange$ Else if In < –WldInRange then: $WldInput = -WldInRange$ Else: $WldInput = In$
Wld input ratio	If |In| ≤ WldInRange: $WldInputRange = \|In\| \times \frac{1}{WldInRange}$ Else: $WldInputRange = 1$

Value	Equation
Wld ratio	If not parabolic mode: $WldRatio = WldInputRatio$ If parabolic mode: $WldRatio = WldInputRatio^2$
Wlds shaping gain	$Wlds = Wld \times (((WldShapeGain - 1) \times WldRatio) + 1)$
Wlds limits	$LowLimit > 0$ $HighLimit = \frac{0.7\pi}{DeltaT}$
Integral output	$ITerm = Kps \times Wlds \times \frac{(WldInput + WldInput_{n-1})}{2} \times DeltaT + ITerm_{n-1}$
Output	$Out = PTerm + ITerm$

Limiting

The instruction stops the ITerm windup based on the state of the hold inputs.

Condition	Action
If HoldHigh is set **and** ITerm > $ITerm_{n-1}$	ITerm = $ITerm_{n-1}$
If HoldLow is set **and** ITerm < $ITerm_{n-1}$	ITerm = $ITerm_{n-1}$

The instruction also stops integrator windup based on the HighLimit and LowLimit values.

Condition	Action
Integrator > HighLimit	Integrator = HighLimit
Integrator > LowLimit	Integrator = LowLimit

The instructions limits the value of Out based on the HighLimit and LowLimit values.

Condition	Action
HighLimit ≤ LowLimit	Out = LowLimit ITerm = LowLimit HighLowLimsInv is set HighAlarm is set LowAlarm is set WldInput = 0

Condition	Action
Out ≥ HighLimit	Out = HighLimit ITerm = ITerm $_{n-1}$ HighAlarm is set
ITerm > HighLimit	ITerm = HighLimit
Out ≤ LowLimit	Out = LowLimit ITerm = ITerm $_{n-1}$ LowAlarm is set
ITerm < LowLimit	ITerm = LowLimit

Affects Math Status Flags

No

Major/Minor Faults

None specific to this instruction. See Common Attributes for operand-related faults.

Execution

Function Block

Condition	Action Taken
Prescan	EnableIn and EnableOut are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is true	EnableIn and EnableOut bit is set to true. The instruction executes.
Instruction first run	Out n-1 = 0 The algorithm used to calculate Out will not be executed.
Instruction first scan	Out n-1 = 0 The algorithm used to calculate Out will not be executed.
Postscan	EnableIn and EnableOut bits are cleared to false.

Structured Text

Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.
Normal Execution	See Tag.EnableIn is true in the Function Block table.

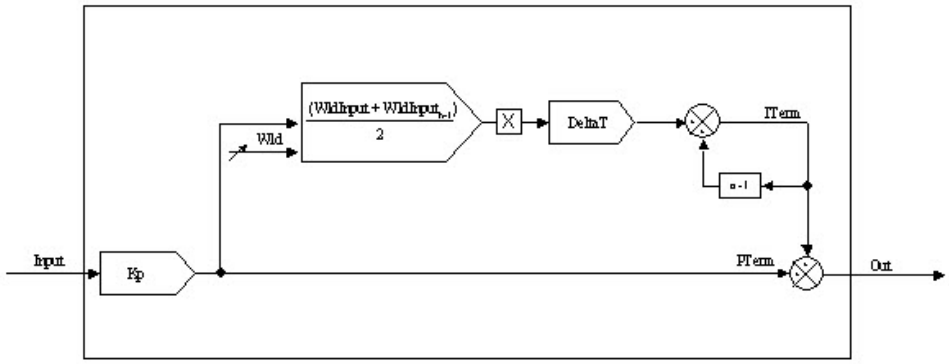
Condition/State	Action Taken
Postscan	See Postscan in the Function Block table.

Example

The PI instruction is a regulating instruction with proportional and integral gain components. The integral gain component is set by the user in radians/sec; this sets the basic frequency response of the PI regulator. The proportional gain sets the overall gain of the block, including the proportional AND integral gain of the block.

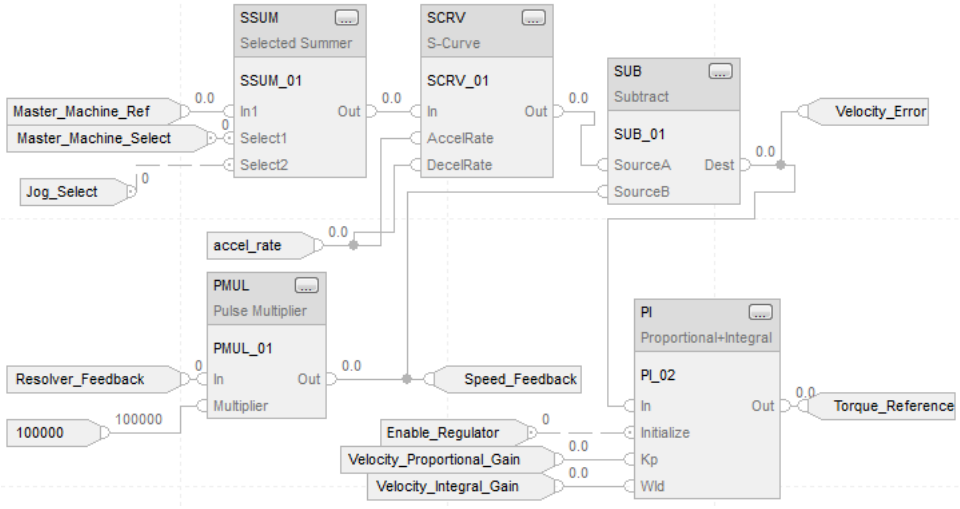
Excluding initialization and holding/clamping functionality, the following diagram shows the PI block's basic regulating loop while in the linear mode.

PI Instruction: Linear Mode



The following example shows the PI instruction used as a velocity regulator. In this example, velocity error is created by subtracting the velocity feedback signal (see the PMUL instruction example) from the system's velocity reference (through the SCRV instruction). Velocity error is driven directly into the PI instruction, which acts on this signal according to the function shown in the diagram above.

Function Block



Structured Text

Reference_Select.In1 := Master_Machine_Ref;

```
Reference_Select.Select1 := Master_Machine_Select;
Reference_Select.In2 := Section_Jog;
Reference_Select.Select2 := Jog_Select;
SSUM(Reference_Select);

S_Curve.In := Reference_Select.Out;
S_Curve.AccelRate := accel_rate;
S_Curve.DecelRate := accel_rate;
SCRV(S_Curve);

PMUL_01.In := Resolver_Feedback;
PMUL_01.WordSize := 12;
PMUL_01.Multiplier := 100000;
PMUL(PMUL_01);

Speed_Feedback := PMUL_01.Out;
Velocity_Error := S_Curve.Out - Speed_Feedback;

PI_01.In := Velocity_Error;
PI_01.Initialize := Enable_Regulator;
PI_01.Kp := Velocity_Proportional_Gain;
PI_01.Wld := Velocity_Integral_Gain;
PI(PI_01);

Torque_Reference := PI_01.Out;
```

Pulse Multiplier (PMUL)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, CompactLogix 5380, ControlLogix 5580, and ControlLogix 5590 controllers.

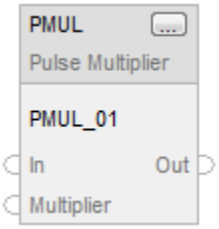
The PMUL instruction provides an interface from a position input module, such as a resolver or encoder feedback module, to the digital system by computing the change in input from one scan to the next. By selecting a specific word size, you configure the PMUL instruction to differentiate through the rollover boundary in a continuous and linear fashion.

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram.

Function Block



Structured Text

PMUL(PMUL_tag);

Operands

Function Block

Operand	Type	Format	Description
PMUL tag	PULSE_MULTIPLIER	Structure	PMUL structure

Structured Text

Operand	Type	Format	Description
PMUL tag	PULSE_MULTIPLIER	Structure	PMUL structure

See Structured Text Syntax for more information on the syntax of expressions within structured text.

PULSE_MULTIPLIER Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
In	DINT	The analog input signal to the instruction. Valid = any DINT Default = 0
Initialize	BOOL	The initialize input. When set, Out is held at 0.0 and all the internal registers are set to 0. On a set to cleared transition of initialize, In_{n-1} = InitialValue (not valid for Absolute mode). When cleared, the instruction executes normally.
InitialValue	DINT	The initial value input. On a set to cleared transition of initialize, In_{n-1} = InitialValue Valid= any DINT Default = 0.
Mode	BOOL	The mode input. Set to enable Relative mode. Clear to enable Absolute mode. Default is set.
WordSize	DINT	The word size in bits. Specify the number of bits to use when

Input Parameter	Data Type	Description
		computing (In_n-In_{n-1}) in Relative mode. WordSize is not used in Absolute mode. When the change in In is greater than 1/2 * $2^{(Wordsize-1)}$, Out changes sign. When WordSize is invalid,Out is held and the instruction sets the appropriate bit in Status. Valid = 2 to 32 Default = 14
Multiplier	DINT	The multiplier. Divide this value by 100,000 to control the ratio of In to Out. If invalid, the instruction limits the value and sets the appropriate bit in Status. Valid = -1,000,000 to 1,000,000 Default = 100,000

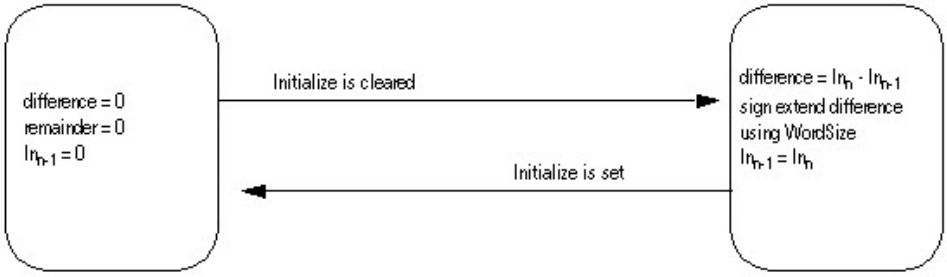
Output Parameter	Data Type	Description
EnableOut	BOOL	Enable output.
Out	REAL	The instruction's Out. If the Out calculation overflows, Out is forced to +/- and the appropriate bit in Status is set. Math status flags are set for this output.
Status	DINT	Status of the function block.
InstructFault (Status.0)	BOOL	The instruction detected one of the following execution errors. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.
WordSizeInv (Status.1)	BOOL	Invalid WordSize value.
OutOverflow (Status.2)	BOOL	The internal output calculation overflowed.
LostPrecision (Status.3)	BOOL	Out < -2^{24} or Out > 2^{24}. When the instruction converts Out from an integer to a real value, data is lost

Output Parameter	Data Type	Description
		if the result is greater than $\|2^{24}\|$ because the REAL data type is limited to 2^{24}.
MultiplierInv (Status.4)	BOOL	Invalid Multiplier value.

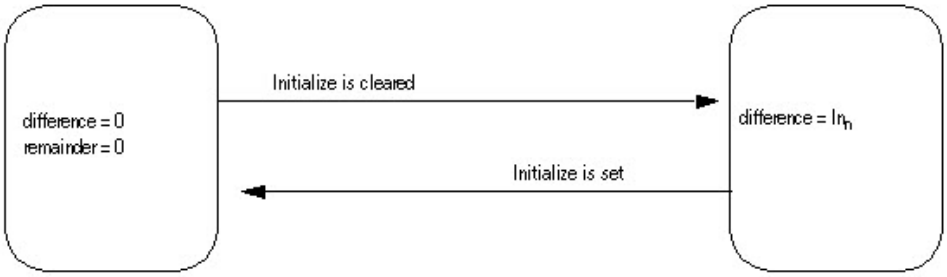
Description

The PMUL instruction operates in Relative or Absolute mode.

In Relative mode, the instruction's output is the differentiation of the input from scan to scan, multiplied by the (Multiplier/100,000). In Relative mode, the instruction saves any remainder after the divide operation in a scan and adds it back in during the next scan. In this manner, position information is not lost over the course of the operation.



In the Absolute mode, the instruction can scale an input, such as position, without losing any information from one scan to the next.



Calculating the Output and Remainder

The PMUL instruction uses these equations to calculate Out in either relative or absolute mode:

Ans = ((DiffInput x Multiplier) + INT_Remainder)

INT_Out = Ans / 100,000

INT_Remainder= Ans - (INT_Out * 100,000)

Out = INT_Out

Affects Math Status Flags

No

Fault conditions

Table 7.

Minor fault occurs when	Fault type	Fault code
Feature is enabled and overflow is detected.	4	4

See Common Attributes on page for operand-related faults.

Execution

Function Block

Condition	Function Block Action
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is true	EnableIn and EnableOut bits are set to true. The instruction executes.
Instruction first run	N/A
Instruction first scan	In_{n-1} = In. Out_{n-1} =0. Reminder = 0.
Postscan	EnableIn and EnableOut bits are cleared to false.

Structured Text

Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.
Normal Execution	See Tag.EnableIn is true in the Function Block table.
Postscan	See Postscan in the Function Block table.

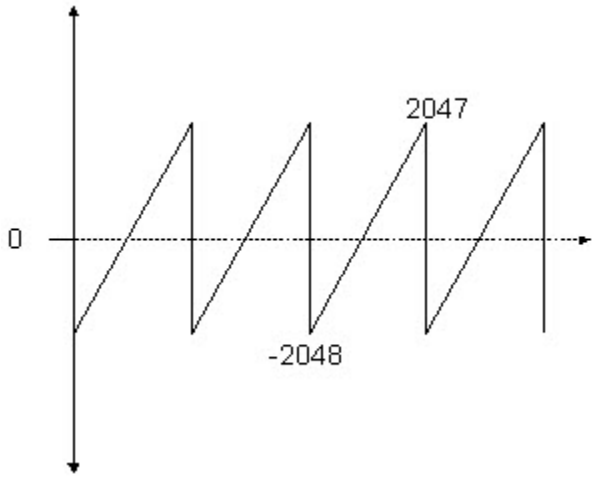
Examples

Example 1

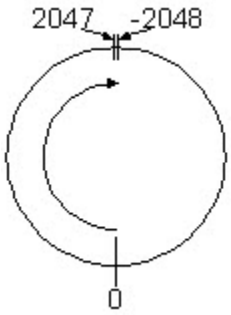
The most common use of the PMUL instruction is in the relative mode of operation. In this mode, the PMUL instruction serves several purposes. First, in the relative mode, the PMUL instruction differentiates the information that it receives at its input from scan to scan. As data is received, the instruction outputs the difference of the input from one scan to the next. This means that if In = 500 at scan "n", and then In = 600 at scan "n+1", Out = 100 at scan "n+1."

Secondly, while in this mode of operation, the PMUL instruction also compensates for "rollover" values of binary data originating from a feedback module. For example, a resolver feedback module may have 12 bits of resolution, represented as a binary value, with sign, ranging from

-2048 to 2047. In terms of raw data coming from the feedback module, the rotation of the feedback device might be represented as shown below:



In this example, as the value of the feedback data moves from 2047 to -2048, the effective change in position is equivalent to a jump of 4095 counts in position. In reality, however, this change in position is only 1 part in 4096 in terms of the rotation of the resolver feedback device. By understanding the true word size of the data that is being input from the feedback module, the PMUL instruction views the data in a rotary fashion as shown in the following diagram:



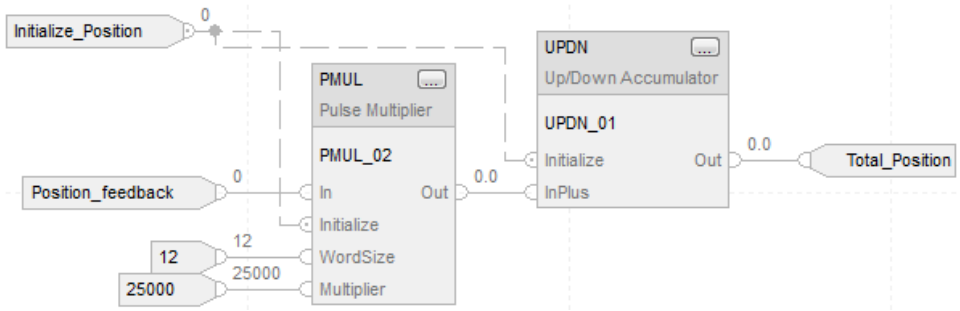
By knowing the word size of the data that is input to this block, the PMUL instruction differentiates an output of 1 count as the input to the block moves from 2047 to -2048, instead of the mathematically calculated 4095.

When applying this block, it is important to note that the feedback data should not change by more than one-half of the word size from one scan to the next, if rotational direction is to be properly differentiated.

In the example above, if the feedback device is moving in a clockwise direction such that at scan 'A' it reads 0 and then scan 'B' it reads -2000, actual change in position is equivalent to +2096 counts in the clockwise direction. However, since these two values are more than one half the words size, (or more than one half the rotation of the physical device,) the PMUL instruction calculates that the feedback device rotated in the opposite direction and returns a value of -2000 instead of +2096.

The third attribute of the pulse multiplier block is that it retains the fractional components from one scan to the next of any remainders that exist as a result of the Multiplier/100,000 scaling factor. As each execution of the block is completed, the remainder from the previous scan is added back into the total of the current value so that all counts or "pulses" are ultimately accounted for and no data is lost in the system. The output of the block, Out always yields a whole number in a floating point data type.

Function Block



Assuming Initial_Position = 0 and Multiplier = 2500 => (25,000/100,000)

Scan	Position_Feedback	PMUL_02.Out	Total_Position
n	0	0	0
n + 1	1	0	0
n + 2	2	0	0
n + 3	3	0	0
n + 4	4	1	1
n + 5	5	0	1

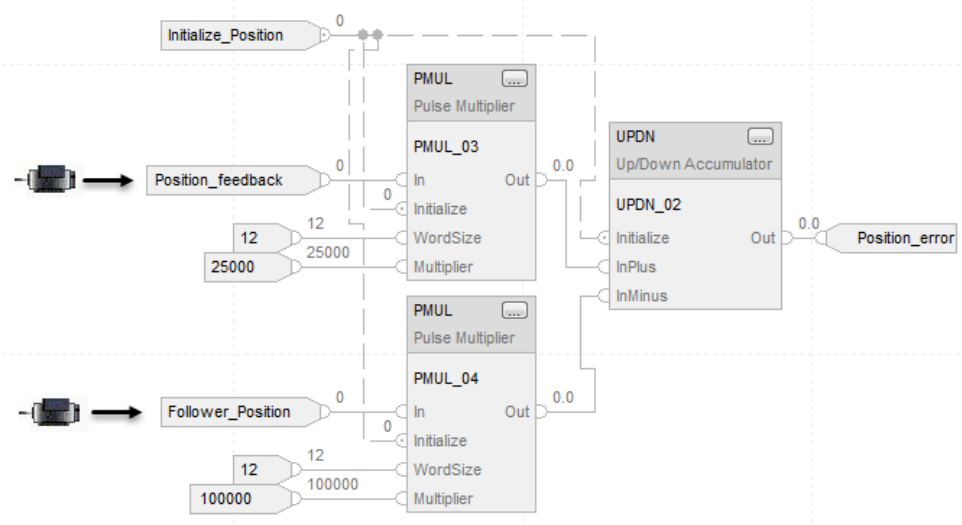
Structured Text

```
MUL_02.In := Position_feedback;
PMUL_02.Initalize := Initialize_Position;
PMUL_02.WordSize := 12;
PMUL_02.Multiplier := 25000;
PMUL(PMUL_02);

UPDN_02.Initialize := Initialize_Position;
UPDN_02.InPlus := PMUL_02.Out;
UPDN(UPDN_02);

Total_Position := UPDN_02.Out;
```

Example 2

In this electronic line shaft application, motor A's feedback acts as a master reference which motor B needs to follow. Motor A's feedback is aliased to "Position_feedback." Motor B's feedback is aliased to "Follower_Position." Due to the multipliers of both instructions being a ratio of 1/4, motor B needs to rotate once for every four revolutions of Motor A in order to maintain an accumulated value of zero in the UPDN accumulator. Any value other than zero on the output of the UPDN instruction is viewed as Position_error and can be regulated and driven back out to motor B in order to maintain a phase-lock between the two motors.

Function Block



Structured Text

```
PMUL_02.In := Position_feedback;
PMUL_02.Initalize := Initialize_Position;
PMUL_02.WordSize := 12;
PMUL_02.Multiplier := 25000;
PMUL(PMUL_02);

PMUL_03.In := Follower_Position;
PMUL_03.Initalize := Initialize_Position;
PMUL_03.WordSize := 12;
PMUL_03.Multiplier := 100000;
PMUL(PMUL_03);

UPDN_02.Initialize := Initialize_Position;
UPDN_02.InPlus := PMUL_02.Out;
UPDN_02.InMinus := PMUL_03.Out;
UPDN(UPDN_02);

Position_error := UPDN_02.Out;
```

S-Curve (SCRV)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, CompactLogix 5380, ControlLogix 5580, and ControlLogix 5590 controllers.

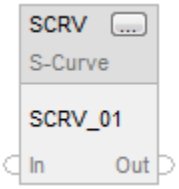
The SCRV instruction performs a ramp function with an added jerk rate. The jerk rate is the maximum rate of change of the rate used to ramp output to input.

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram.

Function Block



Structured Text

SCRV(SCRV_tag);

Operands

Function Block

Operand	Type	Format	Description
SCRV tag	S_CURVE	Structure	SCRV structure

Structured Text

Operand	Type	Format	Description
SCRV tag	S_CURVE	Structure	SCRV structure

See Structured Text Syntax for more information of the syntax of expressions within structured text.

S_CURVE Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
In	REAL	The analog signal input to the instruction. Valid = any float Default = 0.0
Initialize	BOOL	The Initialize input to the instruction. When set, the instruction holds Out = InitialValue Default is cleared.
InitialValue	REAL	Initial value of S-Curve. When Initialize is set, Out = InitialValue. Valid = any float Default = 0.0

Input Parameter	Data Type	Description
AbsAlgRamp	BOOL	Ramp type. If set, the instruction functions as an absolute value ramp. If cleared, the instruction functions as an algebraic ramp. Default is set
AccelRate	REAL	Acceleration rate in input units per second^2. A value of zero prevents Out form accelerating. When AccelRate < 0, the instruction assumes AccelRate = 0 and sets the appropriate bit in Status. Valid = 0.0 to maximum positive float Default = 0.0
DecelRate	REAL	Deceleration rate in input units per second^2. A value of zero prevents Out form decelerating. When the DecelRate < 0, the instruction assumes DecelRate = 0 and sets the appropriate bit in Status. Valid = 0.0 to maximum positive float Default = 0.0
JerkRate	REAL	Jerk rate in input units per second^3. Specifies the maximum rate of change in the acceleration and deceleration rates when ramping output to input. When (JerkRate * DeltaT) S AccelRate or DecelRate, the acceleration and deceleration rates are not bounded. In this situation, the instruction behaves as a ramp function. When JerkRate < 0, the instruction assumes JerkRate = 0 and sets the appropriate bit in Status. Valid = 0.0 to maximum positive float Default = 0.0
HoldMode	BOOL	S-Curve hold mode parameter. This parameter is used with the HoldEnable parameter. If HoldMode is set when HoldEnable is set and Rate = 0, the instruction holds Out constant. In this situation, the instruction holds Out as soon as HoldEnable is set, the JerkRate is ignored, and Out produces a "corner" in its profile. If HoldMode

Input Parameter	Data Type	Description
		is cleared when HoldEnable is set, the instruction uses the JerkRate to bring Out to a constant value. Out is held when Rate = 0. Do not change HoldMode once HoldEnable is set because the instruction will ignore the change. Default is cleared.
HoldEnable	BOOL	S-Curve hold enable parameter. When set, Out is held. When cleared, Out moves from its current value until it equals In. Default is cleared.
TimingMode	DINT	Selects timing execution mode. 0 = periodic mode 1 = oversample mode 2 = real time sampling mode For more information about timing modes, see Function Block Attributes. Valid = 0 to 2 Default = 0
OversampleDT	REAL	Execution time for oversample mode. Valid = 0 to 4194.303 seconds Default = 0
RTSTime	DINT	Module update period for real time sampling mode Valid = 1 to 32,767ms Default = 1
RTSTimeStamp	DINT	Module time stamp value for real time sampling mode. Valid = 0 to 32,767ms Default = 0

Output Parameter	Data Type	Description
EnableOut	BOOL	Enable output.
S_Mode	BOOL	S_Mode Output. When (Jerk * DeltaT) $\leq$ Rate and Rate < Accel or Decel, S_Mode is set. Otherwise, S_Mode is cleared.

Output Parameter	Data Type	Description
Out	REAL	The output of the S-Curve instruction. Math status flags are set for this output.
Rate	REAL	Internal change in the Out in units per second.
DeltaT	REAL	Elapsed time between updates. This is the elapsed time in seconds used by the control algorithm to calculate the process output.
Status	DINT	Status of the function block.
InstructFault (Status.0)	BOOL	The instruction detected one of the following execution errors. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.
AcceRateInv (Status.1)	BOOL	AccelRate is negative.
DecelRateInv (Status.2)	BOOL	DecelRate is negative.
JerkRateInv (Status.3)	BOOL	JerkRate is negative.
TimingModeInv (Status.27)	BOOL	Invalid timing mode. For more information about timing modes, see Function Block Attributes.
RTSMissed (Status.28)	BOOL	Only used in real time sampling mode. Set when ABS | DeltaT - RTSTime | > 1 (.001 second).
RTSTimeInv (Status.29)	BOOL	Invalid RTSTime value.
RTSTimeStampInv (Status.30)	BOOL	Invalid RTSTimeStamp value.
DeltaT (Status.31)	BOOL	Invalid DeltaT value.

Description

The primary requirement of the SCRV instruction is to ensure that the rate never changes by more than the specified jerk rate.

You can configure the SCRV instruction to produce an S-Curve profile or a Ramp profile for a step input.

S-Curve Profile

To produce an S-Curve profile, set JerkRate such that (JerkRate * DeltaT) < AccelRate and/or DecelRate.

In S-Curve profile mode, the SCRV instruction ensures that the rate never changes more than the specified JerkRate. The algorithm used to produce the S-Curve profile is designed to produce a smooth, symmetric S-Curve for a step input. A trapezoidal integration of Out is incorporated to facilitate this. As a result, changes in Rate will be less than JerkRate during portions of the profile.

When a step change occurs on the input, rate is increased to the programmed AccelRate or DecelRate. The AccelRate or DecelRate is maintained until a point at which rate must begin decreasing in order for the output to reach input when rate reaches zero.

In some cases, depending on the values of acceleration, deceleration, and jerk, the acceleration rate or deceleration rate might not be reached before the rate must begin decreasing by jerk rate.

For very small step changes, the SCRV instruction will not attempt to produce an 'S' profile. In this mode the entire step will be output and Rate will reflect the change in output. This behavior will occur if Out = In and the next step change to In can be output with a rate less than or equal to the programmed JerkRate.

The SCRV instruction supports an algebraic ramp and an absolute value ramp. For an algebraic ramp, the acceleration condition is defined by an input that is becoming more positive, and the deceleration condition is defined by an input that is becoming more negative. For an absolute value ramp, the acceleration condition is defined by an input moving away from zero, and the deceleration condition is defined by an input moving towards zero.

Ramp Profile

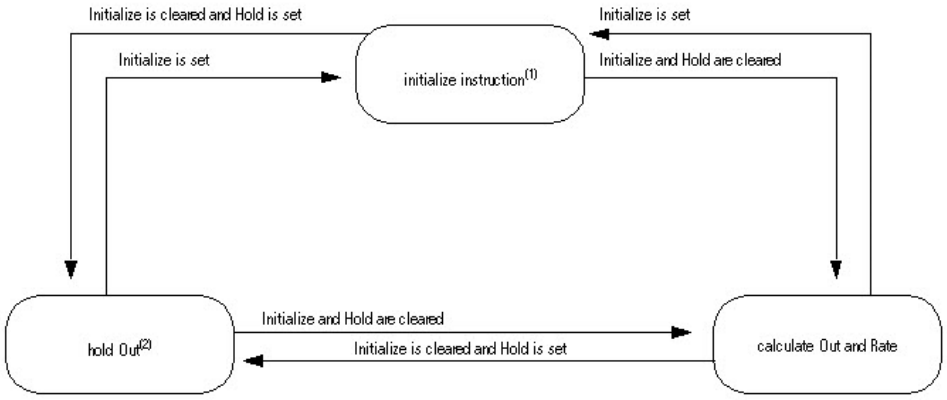
To produce a Ramp profile, set JerkRate such that (JerkRate * DeltaT) ≤ AccelRate and/or

DecelRate.

In Ramp Profile mode, the SCRV instruction always produces a rate of change equal to the programmed AccelRate or DecelRate until the difference between Out and In requires less then AccelRate or DecelRate to reach endpoint.

HoldMode = 0 operates the same as HoldMode = 1. When HoldEnable is set, Out is immediately held and Rate becomes zero.

The following diagram illustrates how the instruction modifies Out.



(1) When Initialize is set, the instruction sets the following:

Outn = InitialValue

Outn-1 = Outn

Raten = 0

Raten-1 = 0

(2) When HoldMode is cleared, Out is moving toward In, and HoldEnable is set, the rate begins decreasing towards zero at the jerk rate. Due to the JerkRate, Out is held at whatever value it had when the rate reached zero. When the Out is finally held constant, it has a value that is different from the value it had the instant that HoldEnable was set.

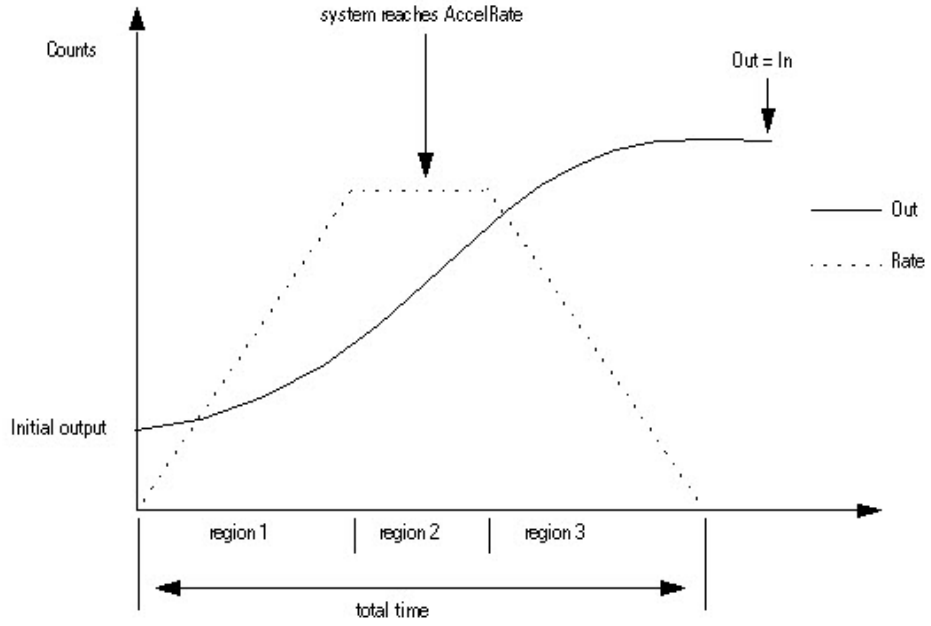
When HoldMode is set, Out is moving toward In, and HoldEnable is set, the rate is immediately set to zero. Out is held at whatever value it had when HoldEnable was set.

Reducing the JerkRate during a transition might cause Out to overshoot the In. If overshoot occurs, it is the result of enforcing the entered JerkRate. You can avoid an overshoot by decreasing JerkRate in small steps while tuning or by changing JerkRate while Out = In (not during a transition).

The time that is required for Out to equal a change in the input is a function of AccelRate, JerkRate, and the difference between In and Out.

Calculating Output and Rate Values

In transition from an initial value to final value, Out goes through three regions. In region 1 and region 3, the rate of change of Out is based on JerkRate. In region 2, the rate of change of Out is based on AccelRate or DecelRate.



The Out is calculated for each region as follows:

$$TotalTime = \frac{FinalOutput - InitialOutput}{AccelRate} + \frac{AccelRate}{JerkRate}$$

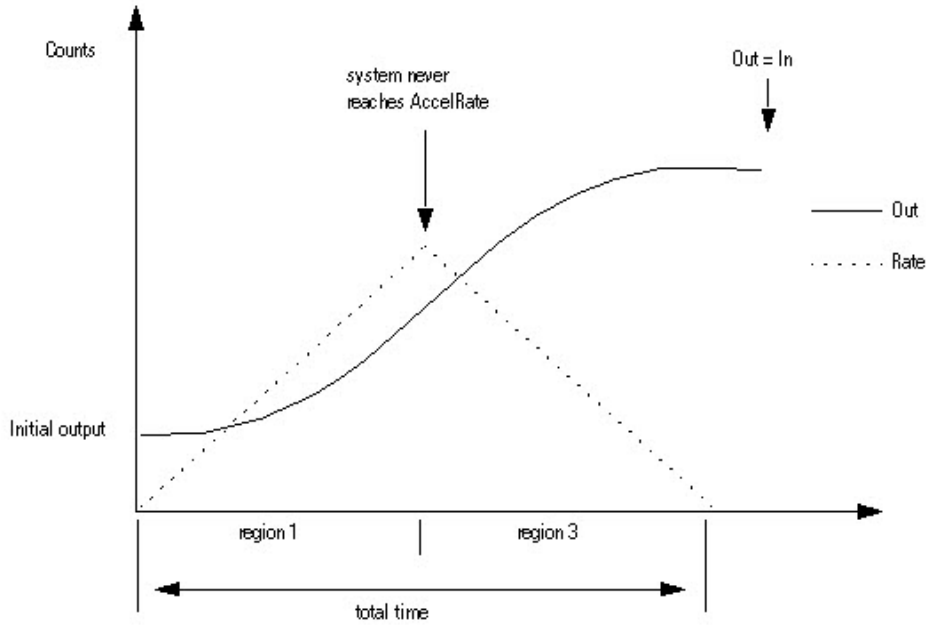
with these equations for each region:

Region	Equation
region 1	$Time_1 = \frac{AccelRate}{JerkRate}$ $Y(Time) = InitialOutput + \frac{1}{2}(JerkRate) \times Time^2$
region 2	$Time_2 = \frac{JerkRate \times (FinalOutput - InitialOutput) - AccelRate^2}{JerkRate \times AccelRate}$ $Y(Time) = InitialOutput + (AccelRate \times Time) - \frac{AccelRate^2}{2 \times JerkRate}$
region 3	$Time_3 = \frac{AccelRate}{JerkRate}$ $Y(Time) = FinalOutput - \frac{1}{2}(JerkRate) \times \left(Time - \frac{FinalOutput - InitiaOutput}{AccelRate} - \frac{AccelRate}{JerkRate}\right)^2$

When:

$$|InitialOutput - FinalOutput| < \frac{AccelRate^2}{JerkRate}$$

the SCRV block does not reach the AccelRate or DecelRate. The Out does the following:



where:

$$TotalTime = 2 \times \sqrt{\frac{|InitialOutput - FinalOutput|}{JerkRate}}$$

Affects Math Status Flags

No

Fault conditions

Table 8.

Minor fault occurs when	Fault type	Fault code
Feature is enabled and overflow is detected.	4	4

See Common Attributes on page for operand-related faults.

Execution

Function Block

Condition	Function Block Action
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is true	EnableIn and EnableOut bit bits are set to true The instruction executes.
Instruction first run	N/A
Instruction first scan	Clear previous scan data.
Postscan	EnableIn and EnableOut bits are cleared to false

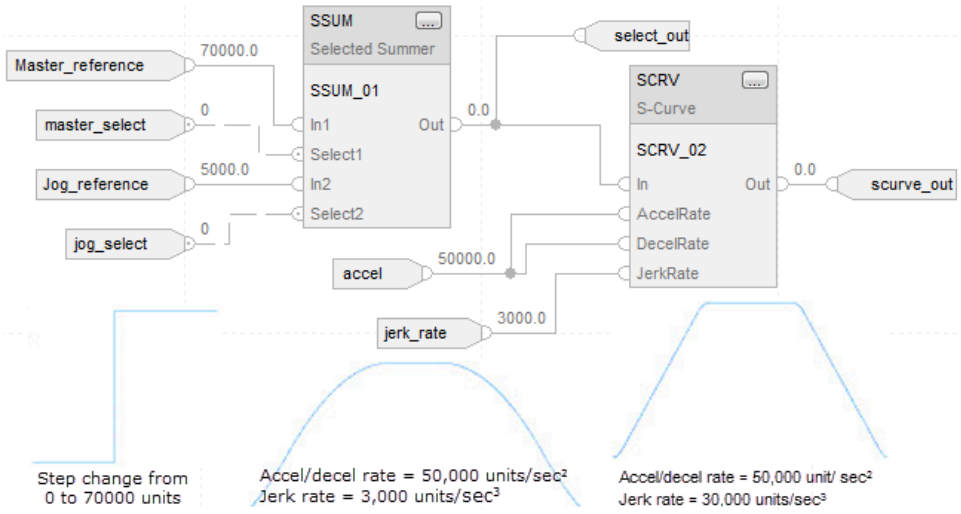
Structured Text

Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.
Normal Execution	See Tag.EnableIn is true in the Function Block table.
Postscan	See Postscan in the Function Block table.

Example

In most coordinated drive applications, a master reference commands line speed for an entire group of drives. As various references are selected, the drives cannot be presented with "step" changes in speed reference because differences in load inertia, motor torque, and tuning would not allow the individual drive sections to react in a coordinated manner. The SCRV instruction is designed to ramp and shape the reference signal to the drive sections so that acceleration, deceleration, and jerk, (derivative of acceleration,) are controlled. This instruction provides a mechanism to allow the reference to the drives to reach the designated reference setpoint in a manner that eliminates excessive forces and excessive impact on connected machinery and equipment.

Function Block



Structured Text

```
SSUM_01.In1 := Master_reference;
SSUM_01.Select1 := master_select;
SSUM_01.In2 := Jog_reference;
SSUM_01.Select2 := jog_select;
SSUM(SSUM_01);

select_out := SSUM_01.Out;

SCRV_01.In := select_out;
SCRV_01.AccelRate := accel;
SCRV_01.DecelRate := accel;
SCRV_01.JerkRate := jerk_rate;
SCRV(SCRV_01);

scurve_out := SCRV_01.Out
```

Second-Order Controller (SOC)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, CompactLogix 5380, ControlLogix 5580, and ControlLogix 5590 controllers.

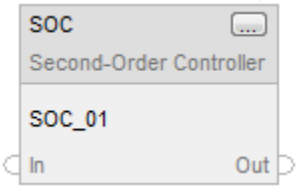
The SOC instruction is designed for use in closed loop control systems in a similar manner to the PI instruction. The SOC instruction provides a gain term, a first order lag, and a second order lead.

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram.

Function Block



Structured Text

SOC(SOC_tag);

Operands

Function Block

Operand	Type	Format	Description
SOC tag	SEC_ORDER_CONTROLLER	Structure	SOC structure

SEC_ORDER_CONTROLLER Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
In	REAL	The analog signal input to the instruction. Valid = any float Default = 0.0
Initialize	BOOL	The instruction initialization command. When set, Out and internal integrator are set equal to the value of InitialValue. Default is cleared.
InitialValue	REAL	The initial value input. When Initialize is set, Out and integrator are set to the value of InitialValue. The value of InitialValue is limited using HighLimit and LowLimit. Valid = any float Default = 0.0
Gain	REAL	The proportional gain for the instruction. If the value is out of range, the instruction limits the value and sets the appropriate bit in Status. Valid = any float > 0.0

Input Parameter	Data Type	Description
		Default = minimum positive float
WLag	REAL	First order lag corner frequency in radians/second. If the value is out of range, the instruction limits the value and sets the appropriate bit in Status. Valid = see the Description section below for valid ranges Default = 0.0
WLead	REAL	Second order lead corner frequency in radians/second. If the value is out of range, the instruction limits the value and sets the appropriate bit in Status. Valid = see the Description section below for valid ranges Default = 0.0
ZetaLead	REAL	Second order lead damping factor. If the value is out of range, the instruction limits the value and sets the appropriate bit in Status. Valid = 0.0 to 10.0 Default = 0.0
HighLimit	REAL	The high limit value. This is the maximum value for Out. If HighLimit ≤ LowLimit, the instruction sets HighAlarm and LowAlarm, sets the appropriate bit in Status, and sets Out = LowLimit. Valid = LowLimit < HighLimit ≤ maximum positive float Default = maximum positive float
LowLimit	REAL	The low limit value. This is the minimum value for Out. If HighLimit ≤ LowLimit, the instruction sets HighAlarm and LowAlarm, sets the appropriate bit in Status, and sets Out = LowLimit. Valid = maximum negative float ≤ LowLimit < HighLimit Default = maximum negative float

Input Parameter	Data Type	Description
HoldHigh	BOOL	The hold high command. When set, the value of the internal integrator is not allowed to increase in value. Default is cleared.
HoldLow	BOOL	The hold low command. When set, the value of the internal integrator is not allowed to decrease in value. Default is cleared.
TimingMode	DINT	Selects timing execution mode. 0 = Periodic mode 1 = Oversample mode 2 = Real time sampling mode For more information about timing modes, see Function Block Attributes. Valid = 0 to 2 Default = 0
OversampleDT	REAL	Execution time for oversample mode. Valid = 0 to 4194.303 seconds Default = 0
RTSTime	DINT	Module update period for real time sampling mode Valid = 1 to 32,767ms Default = 1
RTSTimeStamp	DINT	Module time stamp value for real time sampling mode. Valid = 0 to 32,767ms Default = 0

Output Parameter	Data Type	Description
EnableOut	BOOL	Indicates if instruction is enabled. Cleared to false if Out overflows.
Out	REAL	The calculated output of the algorithm.
HighAlarm	BOOL	The maximum limit alarm indicator. Set when the calculated value for Out ≥HighLimit and the output is clamped at HighLimit.

Output Parameter	Data Type	Description
LowAlarm	BOOL	The minimum limit alarm indicator. Set when the calculated value for Out ≤ LowLimit and the output is clamped at LowLimit.
DeltaT	REAL	Elapsed time between updates. This is the elapsed time in seconds used by the control algorithm to calculate the process output.
Status	DINT	Status of the function block.
InstructFault (Status.0)	BOOL	The instruction detected one of the following execution errors. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.
GainInv (Status.1)	BOOL	Gain < minimum positive float.
WLagInv (Status.2)	BOOL	WLag > maximum or WLag < minimum.
WLeadInv (Status.3)	BOOL	WLead > maximum or WLead < minimum.
ZetaLeadInv (Status.4)	BOOL	ZetaLead > maximum or ZetaLead < minimum.
HighLowLimsInv (Status.5)	BOOL	HighLimit ≤ LowLimit.
TimingModeInv (Status.27)	BOOL	Invalid timing mode. For more information about timing modes, see Function Block Attributes.
RTSMissed (Status.28)	BOOL	Only used in real time sampling mode. Set when ABS | DeltaT - RTSTime | > 1 (.001 second).
RTSTimeInv (Satus.29)	BOOL	Invalid RTSTime value.
RTSTimeStampInv (Status.30)	BOOL	Invalid RTSTimeStamp value.
DeltaT (Status.31)	BOOL	Invalid DeltaT value.

Structured Text

Operand	Type	Format	Description
SOC tag	SEC_ORDER_CONTROLLER	structure	SOC structure

See Structured Text Syntax for more information of the syntax of expressions within structured text.

Description

The SOC instruction provides a gain term, a first order lag, and a second order lead. The frequency of the lag is adjustable and the frequency and damping of the lead is adjustable. The zero pair for the second order lead can be complex (damping is less than unity) or real (damping ≥ to unity). The SOC instruction is designed to execute in a task where the scan rate remains constant.

The SOC instruction uses the following Laplace Transfer equation.

$$H(s) = \frac{K\left(\frac{s^2}{\omega_{Lead}^2} + \frac{2 \times \xi_{Lead} \times s}{\omega_{Lead}} + 1\right)}{s\left(\frac{s}{\omega_{Lag}} + 1\right)}$$

Parameter Limitations

The following SOC parameters have these limits on valid values.

Parameter	Limit
WLead	$LowLimit = \frac{0.00001}{DeltaT}$ $HighLimit = \frac{0.07\pi}{DeltaT}$ where DeltaT is in seconds
WLag	$LowLimit = \frac{0.0000001}{DeltaT}$ $HighLimit = \frac{0.07\pi}{DeltaT}$ where DeltaT is in seconds
ZetaLead	LowLimit = 0.0 HighLimit = 10.0

Whenever the value computed for the output is invalid or NAN, the instruction sets Out = the invalid value. The internal parameters are not updated. In each subsequent scan, the output is computed using the internal parameters from the last scan when the output was valid.

Limiting

The instruction stops wind-up based on state of the Hold inputs.

If:	Then:
HoldHigh is set and Integrator > Integrator $_{n-1}$	Integrator = Integrator$_{n-1}$
HoldLow is set and Integrator < Integrator$_{n-1}$	Integrator = Integrator$_{n-1}$

The instruction also stops integrator windup based on the HighLimit and LowLimit values.

If:	Then:
Integrator > IntegratorHighLimit	Integrator = IntegratorHighLimit
Integrator < IntegratorLowLimit	Integrator = IntegratorLowLimit

where:

$$IntegratorHighLimit = HighLimit \times \frac{Gain \times WLag}{WLead^2}$$

$$IntegratorLowLimit = LowLimit \times \frac{Gain \times WLag}{WLead^2}$$

The instruction also limits the value of Out based on the HighLimit and LowLimit values.

If:	Then:
HighLimit ≤ LowLimit	Out = LowLimit Integrator = IntegratorLowLimit HighLowLimsInv is set HighAlarm is set LowAlarm is set
Out ≥ HighLimit	Out = HighLimit IntegratorLowLimit$_{n-1}$ HighAlarm is set
Out ≤ LowLimit	Out = LowLimit Integrator = Integrator$_{n-1}$ LowAlarm is set

Affects Math Status Flags

No

Major/Minor Faults

Table 9.

Minor fault occurs when	Fault type	Fault code
Feature is enabled and overflow is detected.	4	4

See Common Attributes on page for operand-related faults.

Execution

NOTE: In Structured Text, EnableIn is always true during normal scan. Therefore, if the instruction is in the control path activated by the logic, it will execute. For more details including definitions and general behavior for all Function Block instructions, refer to Publication 1756-RM006G-EN-P, Advanced Process Control and Drives Instructions.

Condition/State	Action Taken
Prescan	No action taken.
Postscan	No action taken.

Conditions that occur only during Normal Scan mode

Condition/State	Action Taken
Instruction first run	The internal parameters and Out are set to 0. Force recalculation of equation coefficients. The primary algorithm is not executed but will validate input parameters.
Instruction first scan	The internal parameters and Out are set to 0. Force recalculation of equation coefficients. The primary algorithm is not executed but will validate input parameters.
EnableIn is false	.EnableOut bit is cleared to false.
EnableIn is true	.EnableOut bit is set to true. The instruction's main algorithm will be executed and outputs will be updated.

Structured Text

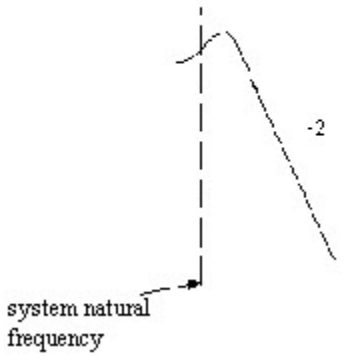
Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.
Normal Execution	See Tag.EnableIn is true in the Function Block table.

Condition/State	Action Taken
Postscan	See Postscan in the Function Block table.

Example

The SOC instruction is a specialized function block that is used in applications where energy is transferred between two sections through a spring-mass system. Typically in these types of applications, the frequency response of the process itself can be characterized as shown in the bode diagram A below:

Diagram A: Process characteristics

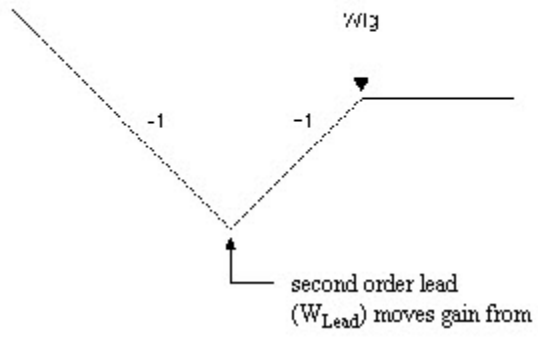


The SOC instruction implements a first order lag filter followed by a PID controller to implement a transfer function with an integration, a second order zero, (lead,) and a first order pole (lag.) With this instruction, PID tuning is simplified because the regulating terms are arranged so that you have WLead and ZLead as inputs to the SOC instruction, rather than Kp, Ki, and Kd values. The transfer function for the SOC instruction is:

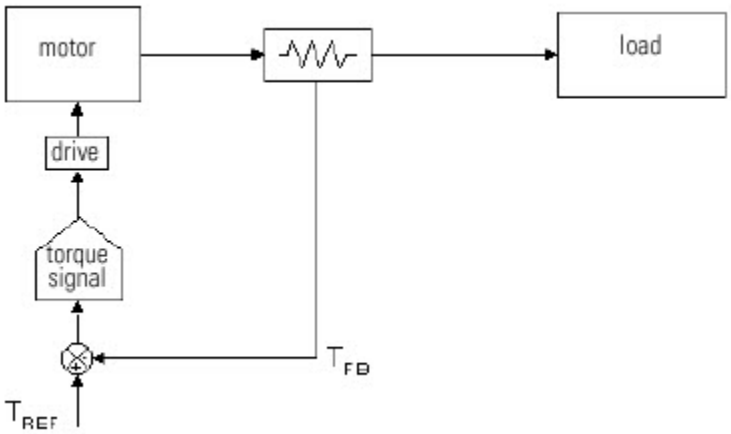
$$H(s) = \frac{K\left(\frac{s^2}{\omega_{Lead}^2} + \frac{2 \times \xi_{Lead} \times s}{\omega_{Lead}} + 1\right)}{s\left(\frac{s}{\omega_{Lag}} + 1\right)}$$

Its corresponding bode diagram is shown in Diagram B below.

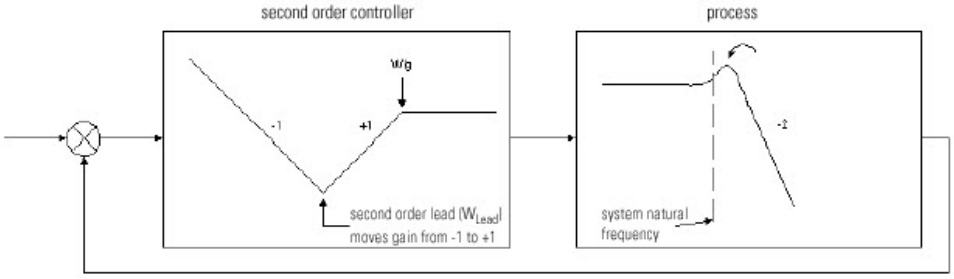
Diagram B: Second order controller



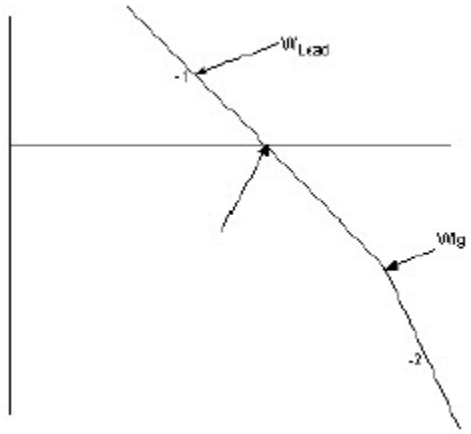
The SOC instruction can be used in a torque or tension regulating application where a load cell or force transducer is used as feedback and the output of the regulating scheme operates directly on the torque (current) minor loop of the drive. In many such applications, the controlled system may be mechanically under-damped and have a natural frequency which is difficult to stabilize as it becomes reflected through the feedback device itself.



Using the SOC instruction, PID tuning is simplified because the regulating terms can be arranged so that you have WLead and ZLead as inputs to the SOC instruction, rather than Kp, Ki, and Kd values. In this manner, the corner frequencies of the controller/regulator are easier to adjust and setup against the real world process. During startup, the natural frequency of the system and the damping factor can be measured empirically or on-site. Afterward, the parameters of the regulator can be adjusted to match the characteristics of the process, allowing more gain and more stable control of the final process.



In the system above, if Wlead is set equal to the system natural frequency, and if Wlag is set substantially above the desired crossover frequency, (> 5 times crossover), the resulting system response would look like the following:



In an actual application, the steps in using and setting up this instruction include:

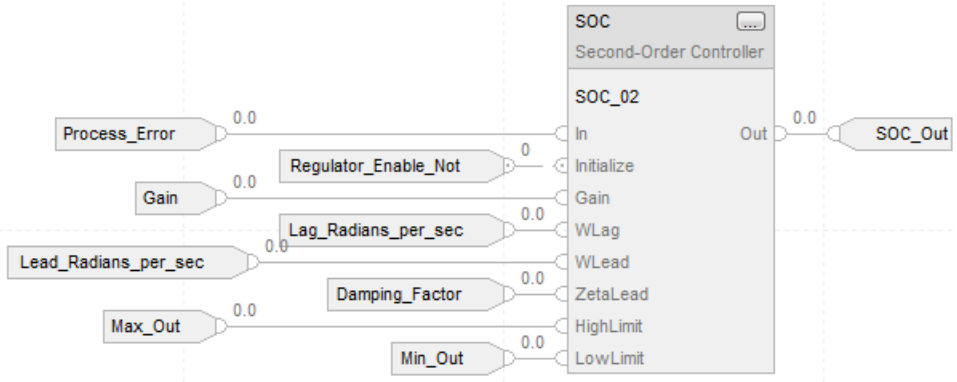
Recognize the type of process that is being controlled. If the system's response to a step function results in a high degree of ringing or can be characterized by the process curve shown above, this block may provide the regulating characteristics required for stable control.

Determine the natural frequency of the system/process. This can may be arrived at empirically - or it might be measured on-site. Adjust WLead so that it corresponds with, or is slightly ahead of, the natural frequency of the process itself.

Tune damping factor, Zlead, so that it cancels out any of the overshoot in the system.

Move WLag out far enough past the system crossover frequency (>5 times) and begin increasing overall Gain to achieve

Function Block



Structured Text

```
SOC_01.In := Process_Error;
SOC_01.Initialize := Regulator_Enable_Not;
SOC_01.Gain := Gain;
SOC_01.WLag := Lag_Radians_per_sec;
SOC_01.WLead := Lead_radians_per_sec;
SOC_01.ZetaLead := Damping_Factor;
```

```
SOC_01.HighLimit := Max_Out;
SOC_01.LowLimit := Min_Out;
SOC(SOC_01);

SOC_Out := SOC_01.Out;
```

Up/Down Accumulator (UPDN)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, CompactLogix 5380, ControlLogix 5580, and ControlLogix 5590 controllers.

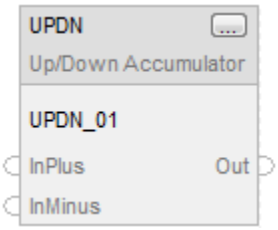
The UPDN instruction adds and subtracts two inputs into an accumulated value.

Available Languages

Ladder Diagram

This instruction is not available for ladder diagram diagram.

Function Block



Structured Text

UPDN(UPDN_tag)

Operands

Function Block

Operand	Type	Format	Description
UPDN tag	UP_DOWN_ACCUM	Structure	UPDN structure

UPDN Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
Initialize	BOOL	The initialize input request for the instruction. When Initialize is set, the instruction sets Out and the internal accumulator to InitialValue.

Input Parameter	Data Type	Description
		Default is cleared.
InitialValue	REAL	The initialize value of the instruction. Valid = any float Default = 0.0
InPlus	REAL	The input added to the accumulator. Valid = any float Default = 0.0
InMinus	REAL	The input subtracted from the accumulator. Valid = any float Default = 0.0
Hold	BOOL	The hold input request for the instruction. When Hold is set and Initialize is cleared, Out is held. Default is cleared.

Output Parameter	Data Type	Description
EnableOut	BOOL	Indicates if instruction is enabled. Cleared to false if Out overflows.
Out	REAL	The output of the instruction.

Structured Text

Operand	Type	Format	Description
UPDN tag	UP_DOWN_ACCUM	Structure	UPDN structure

See Structured Text Syntax for more information of the syntax of expressions within structured text.

Description

The UPDN instruction follows these algorithms.

Condition	Action
Hold is cleared and Initialize is cleared	$AccumValue_n = AccumValue_{n-1} + InPlus - InMinus$ $Out = AccumValue_n$
Hold is set and Initialize is cleared	$AccumValue_n = AccumValue_{n-1}$ $Out = AccumValue_n$
Initialize is set	$AccumValue_n$ = InitialValue

Condition	Action
	Out = $AccumValue_n$

Affects Math Status Flags

No

Fault Conditions

Table 10.

Minor fault occurs when	Fault type	Fault code
Feature is enabled and overflow is detected.	4	4

See Common Attributes on page for operand-related faults.

Execution

Function Block

Condition/State	Action Taken
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is true	EnableIn and EnableOut bits are set to true. The instruction executes.
Instruction first run	Internal accumulator is set to zero.
Instruction first scan	N/A
Postscan	EnableIn and EnableOut bits are cleared to false.

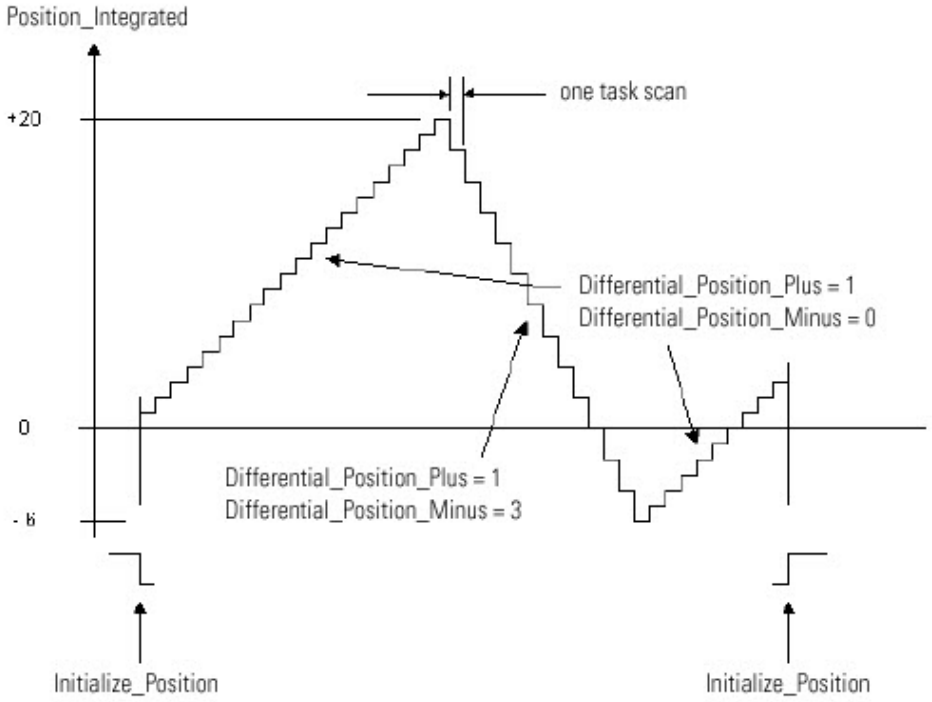
Structured Text

Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.
Normal Execution	See Tag.EnableIn is true in the Function Block table.
Postscan	See Postscan in the Function Block table.

Example

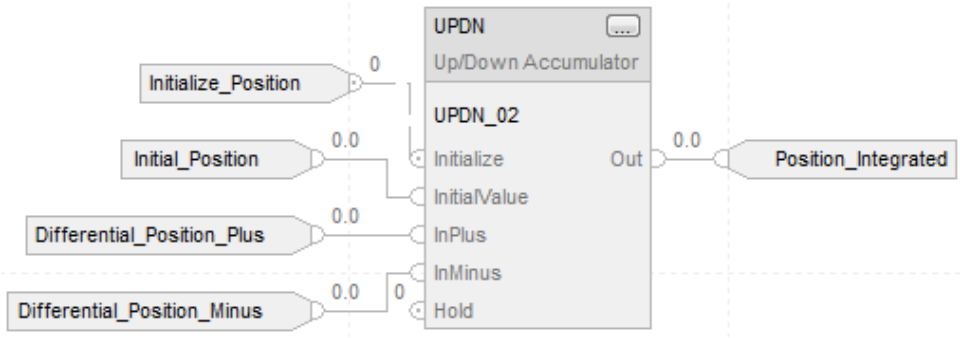
The UPDN instruction integrates counts from one scan to the next. This instruction can be used for simple positioning applications or for other types of applications where simple integration

is required to create an accumulated value from a process's differentiated feedback signal. In the example below, Initial_Position is set to zero, while Differential_Position_Plus and Differential_Position_Minus take varying values over a period of time. With this instruction, InPlus and InMinus could also accept negative values.



Function Block

The derivative instruction calculates the amount of change of a signal over time in per-second units. This instruction is often used in closed loop control to create a feed forward path in the regulator to compensate for processes that have a high degree of inertia.



Structured Text

```
UPDN_01.Initialize := Initialize_Position;
UPDN_01.InitialValue := Initial_Position;
UPDN_01.InPlus := Differential_Position_Plus;
UPDN_01.InMinus := Differential_Position_Minus;
UPDN(UPDN_01);

Position_Integrated := UPDN_01.Out;
```

HMI Button Control (HMIBC)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, Compact GuardLogix 5380, CompactLogix 5380, ControlLogix 5580, GuardLogix 5580, and ControlLogix 5590 controllers.

Use the HMI Button Control (HMIBC) instruction with a PanelView 5500 Human Machine Interface (HMI) to enable operators to initiate machine control operations, such as jogging a motor or enabling a valve, with a high degree of accuracy and determinism. The HMIBC instruction also provides built-in communications diagnostics that permit the instruction to automatically reset if the communications from the controlling HMI become unavailable.

Each Logix controller supports up to 256 HMIBC tags and up to 32 PanelView 5500 HMI's to simultaneously communicate and control the instruction.The HMIBC instruction goes active and enables its output when a PanelView 5500 HMI device initiates a button control operation associated with the instance tag of the instruction.

IMPORTANT: A PanelView 5000 module must be in the project IO tree to use the HMIBC instruction.

To function, the Logix controller I/O configuration must include all of the PanelView 5500 HMIs that need to interact with the HMIBC instruction. Additionally, the application created for each PanelView 5500 HMI must include button actions configured to reference each tag associated with the HMIBC instructions.

ATTENTION: Execute this instruction at least once per scan, and do not jump over.

The HMIBC data type:

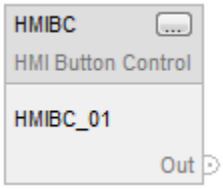
- Is available at Controller and Program scope.
- Is not available within Add-On Instruction scope.
- Is used in a Jump to Subroutine (JSR).
- Cannot be used with input and output program parameters
- Is not available within a safety program.
- Must have an external access value of Read/Write. You are not given the option to choose other external access values.

The HMIBC tag has import and export formats for .L5K, .L5X, and .CSV.

Available Language

Ladder Diagram

Function Block



For the HMIBC tag, use only the Out parameter, and optionally, the ProgFB parameter in Function Block diagrams.

Structured Text

HMIBC (HMIBC tag)

Operands

These operands are located on the instruction.

Operand	Type	Format	Description
HMIBC tag	HMIBC	tag	Goes active when the data bit is set

HMIBC Structure

Input parameter	Data Type	Description
EnableIn	BOOL	Enable input. If false, the instruction does not execute.
Prog FB	BOOL	Program Feedback. This value is not processed by the instruction, but transmitted to all registered HMI devices. The purpose or meaning of this value is user defined. For example, use this to determine if the expected action actually executes when pressing the button and displays that status on the HMI device.

Output parameter	Data Type	Description
EnableOut	BOOL	Indicates if instruction is enabled.
Button State	BOOL	Cleared to false when no registered HMI device buttons are pressed. Set to true when at least one registered HMI button is pressed.

Output parameter	Data Type	Description
		Default value is false.
Out	BOOL	When EnableIn is true: Cleared to false when none of the registered HMI devices buttons are pressed. Set to true when at least one registered HMI button is pressed. When EnableIn is false : Cleared to false Default value is false.

Affects Math Status Flags

No

Major/Minor Faults

None specific to this instruction. See Index through arrays on page 600 for array-indexing faults.

Execution

Ladder Diagram

Condition	Action Taken
Prescan	The rung-condition-out is set to false.
Rung-condition-in is false	The rung-condition-out is set to false.
Rung-condition-in is true	The rung-condition-out is set to true if any HMI device buttons control operation associated with the instruction instance tag are pressed. Otherwise, rung-condition-out is set to false.
Postscan	The rung-condition-out is set to false.

Function Block

Condition/State	Action Taken
Prescan	N/A
Tag.EnableIn is false	The instruction does not execute.
Tag.EnableIn is true	The instruction does not execute.
Instruction first scan	N/A

Condition/State	Action Taken
Instruction first run	N/A
Postscan	N/A

Structured Text

Condition/State	Action Taken
Prescan	The instruction executes.
Normal Execution	The instruction executes.
Postscan	The instruction executes.

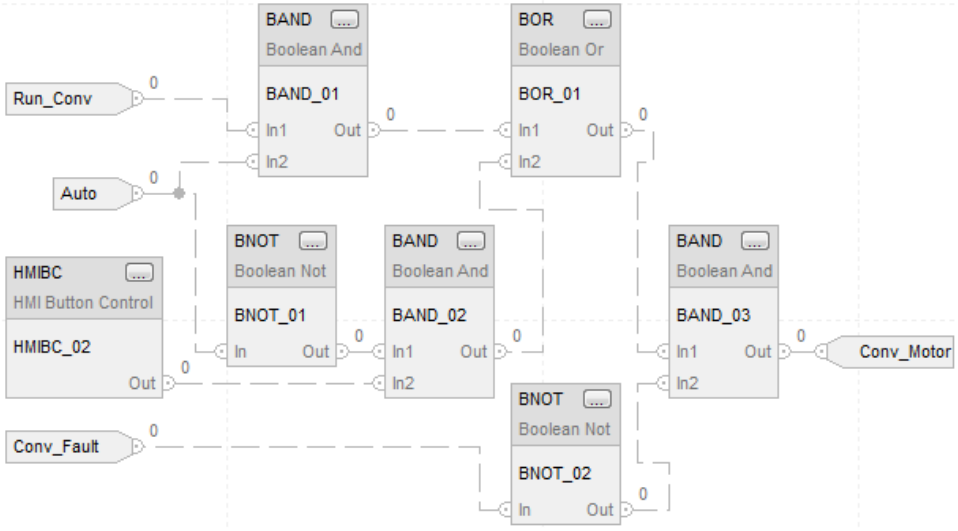
Examples

Ladder Diagram

- An HMIBC instruction is an input instruction and cannot be placed on a rung by itself.
- An HMIBC instruction is highlighted when active.

Function Block

The following example shows the HMIBC instruction as it appears in a function block diagram.



Structured Text

```
HMIBC (HMIBC_Conv);
IF(((Auto AND Run_Conv) Or (NOT Auto AND HMIBC_Conv.Out)) AND NOT Conv_Fault)
THEN Conv_Motor: = 1;
ELSE Conv_Motor : = 0;
END_IF;
```

Filter Instructions

The Filter instructions include these instructions:

Available Instructions

Ladder Diagram

This instruction is not available in Ladder Diagram

Function Block and Structured Text

DERV on page 375	HPF on page 380	LDL2 on page 397	LPF on page 385	NTCH on page 391

If you want to	Use this instruction
Calculate the amount of change of a signal over time in per-second units.	DERV
Filter input frequencies that are below the cutoff frequency.	HPF
Filter with a pole pair and a zero pair.	LDL2
Filter input frequencies that are above the cutoff frequency.	LPF
Filter input frequencies that are at the notch frequency.	NTCH

Related information

Drives Instructions on page 319
Logical and Move Instructions on page 455
Process Control Instructions on page 14
Select/Limit Instructions on page 405
Statistical Instructions on page 436

Derivative (DERV)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, Compact GuardLogix 5380, CompactLogix 5380, ControlLogix 5580, GuardLogix 5580, and ControlLogix 5590 controllers.

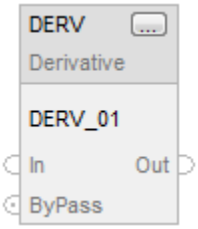
The DERV instruction calculates the amount of change of a signal over time in per-second units.

Available Languages

Ladder Diagram

This instruction is not available for ladder diagram.

Function Block



Structured Text

DERV(DERV_tag);

Operands

Function Block

Operand	Type	Format	Description
DERV tag	DERIVATIVE	structure	DERV structure

DERIVATIVE Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If false, the instruction does not execute and outputs are not updated. Default is true.
In	REAL	The analog signal input to the instruction. Valid = any float Default = 0.0
Gain	REAL	Derivative multiplier Valid = any float Default = 1.0
ByPass	BOOL	Request to bypass the algorithm. When ByPass is true, the instruction sets Out = In. Default is false.
TimingMode	DINT	Selects timing execution mode. 0 = periodic mode 1 = oversampling mode 2 = Real time sampling mode For more information about timing modes, see Function Block Attributes Valid = 0 to 2 Default = 0

Input Parameter	Data Type	Description
OversampleDT	REAL	Execution time for oversample mode. Valid = 0 to 4194.303 seconds Default = 0
RTSTime	DINT	Module update period for real time sampling mode Valid = 1 to 32,767ms Default = 1
RTSTimeStamp	DINT	Module time stamp value for real time sampling mode. Valid = 0 to 32,767ms Default = 0

Output Parameter	Data Type	Description
EnableOut	BOOL	Indicates if instruction is enabled. Cleared to false if Out overflows.
Out	REAL	The calculated output of the algorithm.
DeltaT	REAL	Elapsed time between updates. This is the elapsed time in seconds used by the control algorithm to calculate the process output.
Status	DINT	Status of the function block.
InstructFault (Status.0)	BOOL	The instruction detected one of the following execution errors. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.
TimingModeInv (Status.27)	BOOL	Invalid TimingMode value. For more information about timing modes, see Function Block Attributes.
RTSMissed (Status.28)	BOOL	Only used in real time sampling mode. Set to true when ABS (DeltaT - RTSTime) > 1 millisecond.
RTSTimeInv (Status.29)	BOOL	Invalid RTSTime value.
RTSTimeStampInv (Status.30)	BOOL	Invalid RTSTimeStamp value.
DeltaTInv (Status.31)	BOOL	Invalid DeltaT value.

Structured Text

Operand	Type	Format	Description
DERV tag	DERIVATIVE	structure	DERV structure

See Structured Text Syntax for more information on the syntax of expressions within structured text.

Description

The DERV instruction supports a bypass input that lets you stop calculating the derivative and pass the signal directly to the output.

When Bypass is	The instruction uses this equation
Cleared and DeltaT > 0	$Out = Gain\frac{In_n - In_{n-1}}{DeltaT}$ $In_{n-1} = In_n$ where DeltaT is in seconds
Set	$Out = In_n$ $In_{n-1} = In_n$

Affects Math Status Flags

No

Major/Minor Faults

None specific to this instruction. See Common Attributes for operand-related faults.

Execution

Function Block

Condition/State	Action Taken
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false. Structured Text: NA
Tag.EnableIn is true	EnableIn and EnableOut bits are set to true. The instruction executes.
Instruction first run	N/A
Instruction first scan	Recalculate coefficients.

Condition/State	Action Taken
Postscan	EnableIn and EnableOut bits are cleared to false.

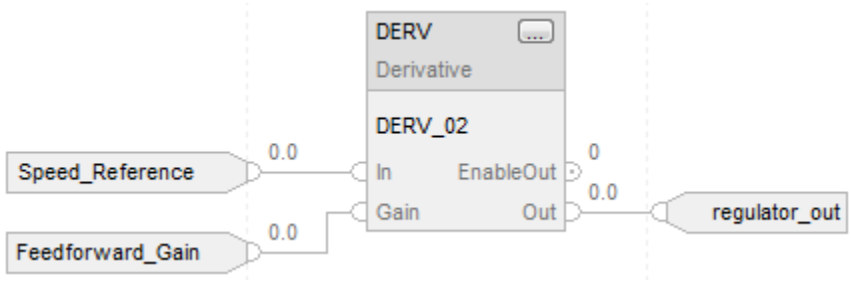
Structured Text

Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.
Normal Execution	See Tag.EnableIn is true in the Function Block table.
Postscan	See Postscan in the Function Block table.

Examples

This example is the minimal legal programming of the DERV function block and is only used to show the neutral text and generated code for this instruction. This is for internal purposes only and is not a testable case.

Function Block



Structured Text

```
DERV_01.In := Speed_Reference;
DERV_01.Gain := Feedforward_Gain;
DERV(DERV_01);

PI_01.In := Speed_Reference - Speed_feedback;
PI_01.Kp := Proportional_Gain;
PI_01.Wld := Integral_Gain;
PI(PI_01);

regulator_out := DERV_01.Out + PI_01.Out;
```

Related information

Function Block Attributes on page 546

Common Attributes on page

Structured Text Syntax on page 563

High Pass Filter (HPF)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, Compact GuardLogix 5380, CompactLogix 5380, ControlLogix 5580, GuardLogix 5580, and ControlLogix 5590 controllers.

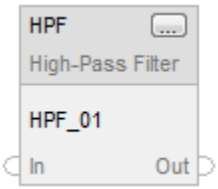
The HPF instruction provides a filter to attenuate input frequencies that are below the cutoff frequency.

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram logic.

Function Block



Structured Text

HPF(HPF_tag);

Operands

Function Block

Operand	Type	Format	Description
HPF tag	FILTER_HIGH_PASS	structure	HPF structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If false, the instruction does not execute and outputs are not updated. Default is true.
In	REAL	The analog signal input to the instruction. Valid = any float Default = 0.0

Input Parameter	Data Type	Description
Initialize	BOOL	Request to initialize filter control algorithm. When true, the instruction sets Out = In. Default is false.
WLead	REAL	The lead frequency in radians/second. If WLead < minimum or WLead > maximum, the instruction sets the appropriate bit in Status and limits WLead. Valid = see Description section below for valid ranges. Default = 0.0
Order	REAL	Order of the filter. Order controls the sharpness of the cutoff. If Order is invalid, the instruction sets the appropriate bit in Status and uses Order = 1. Valid = 1 to 3 Default = 1
OversampleDT	REAL	Execution time for oversample mode. Valid = 0 to 4194.303 seconds Default = 0
RTSTime	DINT	Module update period for real time sampling mode Valid = 1 to 32,767ms Default = 1
RTSTimeStamp	DINT	Module time stamp value for real time sampling mode. Valid = 0 to 32,767ms Default = 0

Output Parameter	Data Type	Description
EnableOut	BOOL	Indicates if instruction is enabled. Cleared to false if Out overflows.
Out	REAL	The calculated output of the algorithm.
DeltaT	REAL	Elapsed time between updates. This is the elapsed time in seconds used by the control algorithm to calculate the process output.
Status	DINT	Status of the function block.

Output Parameter	Data Type	Description
InstructFault (Status.0)	BOOL	The instruction detected one of the following execution errors. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.
WLeadInv (Status.1)	BOOL	WLead < minimum value or WLead > maximum value.
OrderInv (Status.2)	BOOL	Invalid Order value.
TimingModeInv (Status.27)	BOOL	Invalid TimingMode value. For more information about timing modes, see Function Block Attributes.
RTSMissed (Status.28)	BOOL	Only used in real time sampling mode. Set to true when ABS (DeltaT - RTSTime) > 1 millisecond.
RTSTimeInv (Status.29)	BOOL	Invalid RTSTime value.
RTSTimeStampInv (Status.30)	BOOL	Invalid RTSTimeStamp value.
DeltaTInv (Status.31)	BOOL	Invalid DeltaT value.

Structured Text

Operand	Type	Format	Description
HPF tag	FILTER_HIGH_PASS	Structure	HPF structure

See Structured Text Syntax for more information on the syntax of expressions within structured text.

Description

The HPF instruction uses the Order parameter to control the sharpness of the cutoff. The HPF instruction is designed to execute in a task where the scan rate remains constant.

The HPF instruction uses these equations:

When:	The instruction uses this transfer function:
Order = 1	$\frac{s}{s+\omega}$
Order = 2	$\frac{s^2}{s^2+\sqrt{2}\times s\times \omega+\omega^2}$

When:	The instruction uses this transfer function:
Order = 3	$\frac{s^3}{s^3+(2\times s^2\times \omega)+2\times s\times \omega^2+\omega^3}$

with these parameter limits (where DeltaT is in seconds):

Parameter	Limitations
WLead first order LowLimit	$\frac{0.0000001}{DeltaT}$
WLead second order LowLimit	$\frac{0.00001}{DeltaT}$
WLead third order LowLimit	$\frac{0.001}{DeltaT}$
HighLimit	$\frac{0.7\pi}{DeltaT}$

Whenever the value computed for the output is invalid, NAN, or ± INF, the instruction sets Out = the invalid value. When the value computed for the output becomes valid, the instruction initializes the internal parameters and sets Out = In.

Affects Math Status Flags

No

Major/Minor Faults

None specific to this instruction. See Common Attributes for operand-related faults.

Execution

Function Block

Condition/State	Action Taken
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is true	EnableIn and EnableOut are set to true. The instruction executes.
Instruction first run	N/A

Condition/State	Action Taken
Instruction first scan	Recalculate coefficients.
Postscan	EnableIn and EnableOut bits are cleared to false.

Structured Text

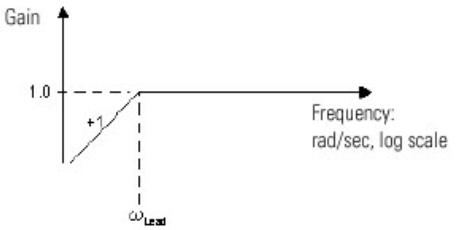
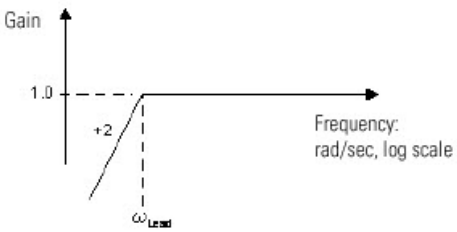
Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.
Normal Execution	See Tag.EnableIn is true in the Function Block table.
Postscan	See Postscan in the Function Block table.

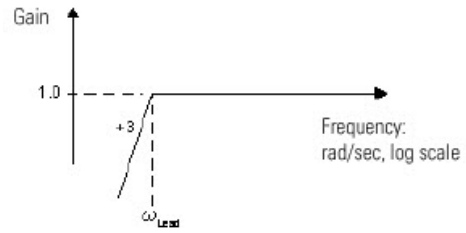
Example

The HPF instruction attenuates signals that occur below the configured cutoff frequency. This instruction is typically used to filter low frequency "noise" or disturbances that originate from either electrical or mechanical sources. You can select a specific order of the filter to achieve various degrees of attenuation. Note that higher orders increase the execution time for the filter instruction.

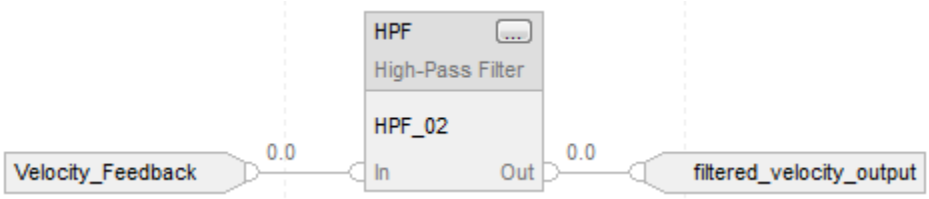
The following graphs illustrate the effect of the various orders of the filter for a given cutoff frequency. For each graph, ideal asymptotic approximations are given with gain and frequency in logarithmic scales. The actual response of the filter approaches these curves but does not exactly match these curves.

This example is the minimal legal programming of the HPF function block and is only used to show the neutral text and generated code for this instruction. This is for internal purposes only and is not a testable case.

Filter	Graph
1st order filter	
2nd order filter	

Filter	Graph
3rd order filter	

Function Block



Structured Text

```
HPF_01.In := Velocity_Feedback;
HPF_01.WLead := Cutoff_frequency;
HPF_01.Order := 2;

HPF(HPF_01);

filtered_velocity_output := HPF_01.Out
```

Related information

Common Attributes on page

Structured Text Syntax on page 563

Low Pass Filter (LPF)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, Compact GuardLogix 5380, CompactLogix 5380, ControlLogix 5580, GuardLogix 5580, and ControlLogix 5590 controllers.

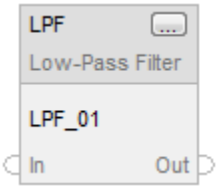
The LPF instruction provides a filter to attenuate input frequencies that are above the cutoff frequency.

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram logic.

Function Block



Structured Text

LPF(LPF_tag);

Operands

Function Block

Operand	Type	Format	Description
LPF tag	FILTER_LOW_PASS	Structure	LPF structure

FILTER_LOW_PASS Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If false, the instruction does not execute and outputs are not updated. Default is true.
In	REAL	The analog signal input to the instruction. Valid = any float Default = 0.0
Initialize	BOOL	Request to initialize filter control algorithm. When true, the instruction sets Out = In. Default is false.
WLag	REAL	The lag frequency in radians/second. If WLag < minimum or WLag > maximum, the instruction sets the appropriate bit in Status and limits WLag. Valid = see Description section below for valid ranges Default = 0.0
Order	REAL	Order of the filter. Order controls the sharpness of the cutoff. If Order is invalid, the instruction sets the appropriate bit in Status and uses Order = 1. Valid = 1 to 3

Input Parameter	Data Type	Description
		Default = 1
TimingMode	DINT	Selects timing execution mode. 0 = Period mode 1 = Oversample mode 2 = Real-time sampling mode For more information about timing modes, see Function Block Attributes. Valid = 0 to 2 Default = 0
OversampleDT	REAL	Execution time for oversample mode. Valid = 0 to 4194.303 seconds Default = 0
RTSTime	DINT	Module update period for real time sampling mode Valid = 1 to 32,767ms Default = 1
RTSTimeStamp	DINT	Module time stamp value for real time sampling mode. Valid = 0 to 32,767ms Default = 0

Output Parameter	Data Type	Description
EnableOut	BOOL	Indicates if instruction is enabled Cleared to false if Out overflows.
Out	REAL	The calculated output of the algorithm.
DeltaT	REAL	Elapsed time between updates. This is the elapsed time in seconds used by the control algorithm to calculate the process output.
Status	DINT	Status of the function block.
InstructFault (Status.0)	BOOL	The instruction detected one of the following execution errors. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.
WLagInv (Status.1)	BOOL	WLag < minimum value or WLag > maximum value.

Output Parameter	Data Type	Description
OrderInv (Status.2)	BOOL	Invalid Order value.
TimingModeInv (Status.27)	BOOL	Invalid TimingMode value.
RTSMissed (Status.28)	BOOL	Only used in real time sampling mode. Set to true when ABS(DeltaT – RTSTime) > 1 millisecond.
RTSTimeInv (Status.29)	BOOL	Invalid RTSTime value.
RTSTimeStampInv (Status.30)	BOOL	Invalid RTSTimeStamp value.
DeltaTInv (Status.31)	BOOL	Invalid DeltaT value.

Structured Text

Operand	Type	Format	Description
LPF tag	FILTER_LOW_PASS	structure	LPF structure

See Structured Text Syntax for more information on the syntax of expressions within structured text.

Description

The LPF instruction uses the Order parameter to control the sharpness of the cutoff. The LPF instruction is designed to execute in a task where the scan rate remains constant.

The LPF instruction uses these equations:

When:	The instruction uses this Laplace transfer function:
Order = 1	$\frac{\omega}{s+\omega}$
Order = 2	$\frac{\omega^2}{s^2+\sqrt{2}\times s\times \omega+\omega^2}$
Order = 3	$\frac{\omega^3}{s^3+(2\times s^2\times \omega)+(2\times s\times \omega^2)\times \omega^3}$

with these parameters limits (where DeltaT is in seconds):

Parameter	Limitations
WLag first order LowLimit	$\frac{0.0000001}{DeltaT}$

Parameter	Limitations
WLag second order LowLimit	$\frac{0.00001}{DeltaT}$
WLag third order LowLimit	$\frac{0.001}{DeltaT}$
HighLimit	$\frac{0.7\pi}{DeltaT}$

Whenever the value computed for the output is invalid, NAN, or ± INF, the instruction sets Out = the invalid value. When the value computed for the output becomes valid, the instruction initializes the internal parameters and sets Out = In.

Affects Math Status Flags

Controllers	Affects Math Status Flags
ControlLogix 5580	No
CompactLogix 5370, ControlLogix 5570	Yes for the output

Major/Minor Faults

None specific to this instruction. See Common Attributes for operand-related faults.

Execution

Function Block

Condition/State	Action Taken
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is true	EnableIn and EnableOut bits are set to true. The instruction executes.
Instruction first run	N/A
Instruction first scan	Recalculate coefficients.
Postscan	EnableIn and EnableOut bits are cleared to false.

Structured Text

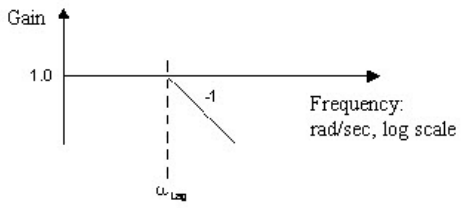
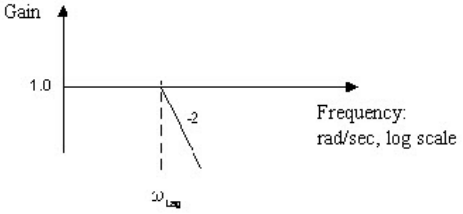
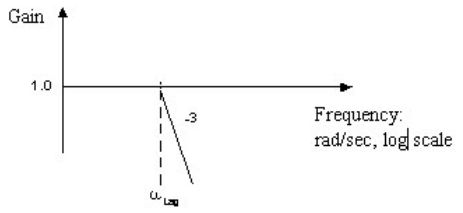
Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.
Normal Execution	See Tag.EnableIn is true in the Function Block table.
Postscan	See Postscan in the Function Block table.

Example

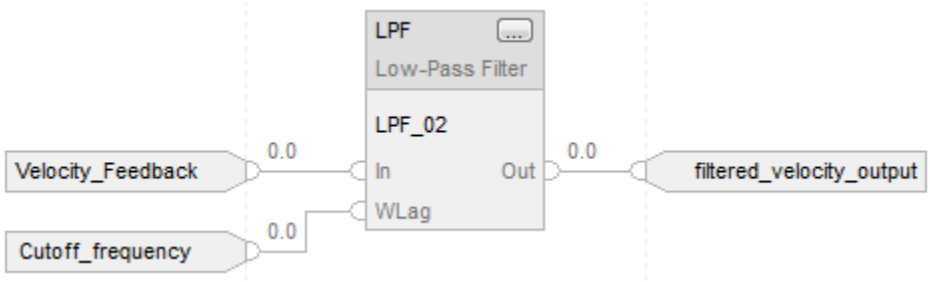
The LPF instruction attenuates signals that occur above the configured cutoff frequency. This instruction is typically used to filter out high frequency "noise" or disturbances that originate from either electrical or mechanical sources. You can select a specific order of the filter to achieve various degrees of attenuation. Note that higher orders increase the execution time for the instruction.

The following graphs illustrate the effect of the various orders of the filter for a given cutoff frequency. For each graph, ideal asymptotic approximations are given with gain and frequency in logarithmic scales. The actual response of the filter approaches these curves but does not exactly match these curves.

This example is the minimal legal programming of the LPF function block and is only used to show the neutral text and generated code for this instruction. This is for internal purposes only and is not a testable case.

Filter	Graph
1st order filter	
2nd order filter	
3rd order filter	

Function Block



Structured Text

```
LPF_01.In := Velocity_Feedback;
LPF_01.WLag := Cutoff_frequency;

LPF(LPF_01);

filtered_velocity_output := LPF_01.Out;
```

Related information

Function Block Attributes on page 546

Common Attributes on page

Structured Text Syntax on page 563

Notch Filter (NTCH)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, Compact GuardLogix 5380, CompactLogix 5380, ControlLogix 5580, GuardLogix 5580, and ControlLogix 5590 controllers.

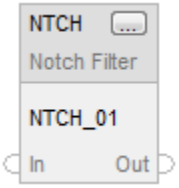
The NTCH instruction provides a filter to attenuate input frequencies that are at the notch frequency.

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram logic.

Function Block



Structured Text

```
NTCH(NTCH_tag);
```

Operands

Function Block

Operand	Type	Format	Description
NTCH tag	FILTER_NOTCH	Structure	NTCH structure

FILTER_NOTCH Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If false, the instruction does not execute and outputs are not updated. Default is true.
In	REAL	The analog signal input to the instruction. Valid = any float Default = 0.0
Initialize	BOOL	Request to initialize filter control algorithm. When true, the instruction sets Out = In. Default is false.
WNotch	REAL	The filter center frequency in radians/second. If WNotch < minimum or WNotch > maximum, the instruction sets the appropriate bit in status and limits WNotch. Valid = see Description section below for valid ranges Default = maximum positive float
QFactor	REAL	Controls the width and depth ratio. Set QFactor = 1 / (2*desired damping factor). If QFactor < minimum or QFactor > maximum value, the instruction sets the appropriate bit in Status and limits QFactor. Valid = 0.5 to 100.0 Default = 0.5
Order	REAL	Order of the filter. Order controls the sharpness of the cutoff. If Order is invalid, the instruction sets the appropriate bit in Status and uses Order = 2. Valid = 2 or 4 Default = 2
TimingMode	DINT	Selects timing execution mode.

Input Parameter	Data Type	Description
		0 = Periodic mode 1 = Oversample mode 2 = Real time sampling mode2 For more information about timing modes, see Function Block Attributes. Valid = 0 to 2 Default = 0
OversampleDT	REAL	Execution time for oversample mode. Valid = 0 to 4194.303 seconds Default = 0
RTSTime	DINT	Module update period for real time sampling mode Valid = 1 to 32,767ms Default = 1
RTSTimeStamp	DINT	Module time stamp value for real time sampling mode. Valid = 0 to 32,767ms Default = 0

Output Parameter	Data Type	Description
EnableOut	BOOL	Indicates if instruction is enabled. Cleared to false if Out overflows.
Out	REAL	The calculated output of the algorithm.
DeltaT	REAL	Elapsed time between updates. This is the elapsed time in seconds used by the control algorithm to calculate the process output.
Status	DINT	Status of the function block.
InstructFault (Status.0)	BOOL	The instruction detected one of the following execution errors. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.
WNotchInv (Status.1)	BOOL	WNotch < minimum or WNotch > maximum
QFactorInv (Status.2)	BOOL	QFactor < minimum or QFactor > maximum

Output Parameter	Data Type	Description
OrderInv (Status.3)	BOOL	Invalid Order value.
TimingModeInv (Status.27)	BOOL	Invalid TimingMode value. For more information about timing modes, see Function Block Attributes.
RTSMissed (Status.28)	BOOL	Only used in real time sampling mode. Set to true when ABS (DeltaT - RTSTime) > 1 millisecond.
RTSTimeInv (Status.29)	BOOL	Invalid RTSTime value.
RTSTimeStampInv (Status.30)	BOOL	Invalid RTSTimeStamp value.
DeltaTInv (Status.31)	BOOL	Invalid DeltaT value.

Structured Text

Operand	Type	Format	Description
NTCH tag	FILTER_NOTCH	structure	NTCH structure

See Structured Text Syntax for more information on the syntax of expressions within structured text.

Description

The NTCH instruction uses the Order parameter to control the sharpness of the cutoff. The QFactor parameter controls the width and the depth ratio of the notch. The NTCH instruction is designed to execute in a task where the scan rate remains constant.

The NTCH instruction uses this equation:

$$\frac{\left(s^2+\omega^2\right)^i}{\left(s^2+s\times\frac{\omega}{Q}+\omega^2\right)^i}$$

where i is the Order operator with these parameters limits (where DeltaT is in seconds):

Parameter	Limitations
WNotch second order LowLimit	$\frac{0.0000001}{DeltaT}$
WNotch fourth order LowLimit	$\frac{0.001}{DeltaT}$

Parameter	Limitations
HighLimit	$\frac{0.7\pi}{DeltaT}$
QFactor	LowLimit = 0.5 HighLimit = 100.0

Whenever the value computed for the output is invalid, NAN, or ± INF, the instruction sets Out = the invalid value. When the value computed for the output becomes valid, the instruction initializes the internal parameters and sets Out = In.

Affects Math Status Flags

No

Major/Minor Faults

None specific to this instruction. See Common Attributes for operand-related faults.

Execution

Function Block

Condition/State	Action Taken
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is true	EnableIn and EnableOut bits are set to true. The instruction executes.
Instruction first run	N/A
Instruction first scan	Recalculate coefficients.
Postscan	EnableIn and EnableOut bits are cleared to false.

Structured Text

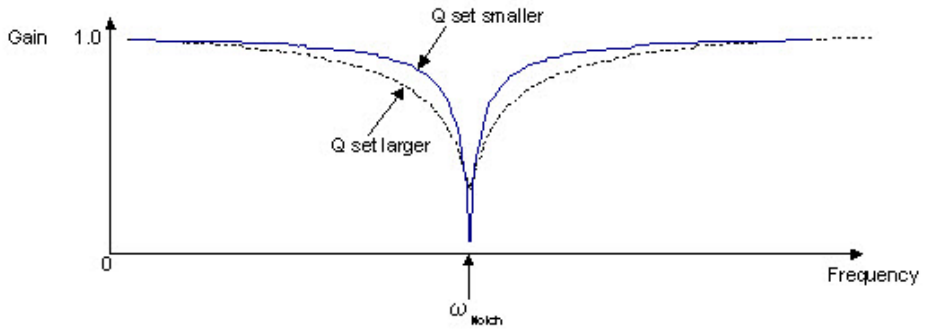
Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.
Normal Execution	See Tag.EnableIn is true in the Function Block table.
Postscan	See Postscan in the Function Block table.

Example

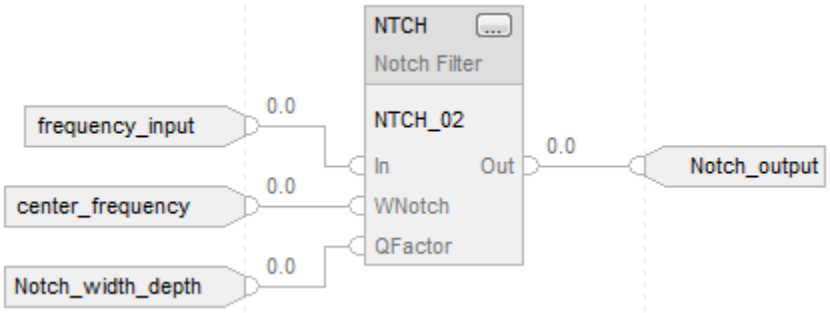
The NTCH instruction attenuates a specific resonance frequency. Typically, these resonance frequencies are directly in the range of response being regulated by the closed loop control system. Often, they are generated by loose mechanical linkages that cause backlash and vibration in the system. Although the best solution is to correct the mechanical compliance in the machinery, the notch filter can be used to soften the effects of these signals in the closed loop regulating scheme.

The following diagram shows the ideal gain curve over a frequency range for a specific center frequency and Q factor. As increases, the notch becomes wider and shallower. As decreases; the notch becomes deeper and narrower. The instruction may be set for an order of 2 or an order of 4. Higher orders take more execution time.

This example is the minimal legal programming of the NTCH function block and is only used to show the neutral text and generated code for this instruction. This is for internal purposes only and is not a testable case.



Function Block



Structured Text

```
NTCH_01.In := frequency_input;
NTCH_01.WNotch := center_frequency;
NTCH_01.QFactor := Notch_width_depth;
NTCH(NTCH_01);

Notch_output := NTCH_01.Out;
```

Related information

Common Attributes on page

Structured Text Syntax on page 563

Second-Order Lead Lag (LDL2)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, Compact GuardLogix 5380, CompactLogix 5380, ControlLogix 5580, GuardLogix 5580, and ControlLogix 5590 controllers.

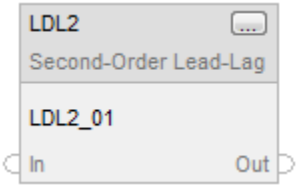
The LDL2 instruction provides a filter with a pole pair and a zero pair. The frequency and damping of the pole and zero pairs are adjustable. The pole or zero pairs can be either complex (damping less than unity) or real (damping greater than or equal to unity).

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram logic.

Function Block



Structured Text

LDL2(LDL2_tag);

Operands

Function Block

Operand	Type	Format	Description
LDL2 tag	LEAD_LAG_SEC_ORDER	Structure	LDL2 structure

LEAD_LAG_SEC_ORDER Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If false, the instruction does not execute and outputs are not updated. Default is true.
In	REAL	The analog signal input to the instruction. Valid = any float Default = 0.0

Input Parameter	Data Type	Description
Initialize	BOOL	Request to initialize filter control algorithm. When true, the instruction sets Out = In. Default is cleared.
WLead	REAL	The lead corner frequency in radians/second. If WLead < minimum or WLead > maximum, the instruction sets the appropriate bit to true in Status and limits WLead. If the WLag:WLead ratio > maximum ratio, the instruction sets the appropriate bit in Status to true and limits WLag Valid = see Description section below for valid ranges. Default = 0.0
WLag	REAL	The lag corner frequency in radians/second. If WLag < minimum or WLag > maximum, the instruction sets the appropriate bit to true in Status and limits WLag. If the WLag:WLead ratio > maximum ratio, the instruction sets the appropriate bit to true in Status and limits WLag. Valid = see Description section below for valid ranges Default = 0.0
ZetaLead	REAL	Second order lead damping factor. Only used when Order = 2. If ZetaLead < minimum or ZetaLead > maximum, the instruction sets the appropriate bit to true in Status and limits ZetaLead. Valid = 0.0 to 4.0 Default = 0.0
ZetaLag	REAL	Second order lag-damping factor. Only used when Order = 2. If ZetaLag < minimum or ZetaLag > maximum, the instruction sets the appropriate bit to true in Status and limits ZetaLag. Valid = 0.05 to 4.0 Default = 0.05
Order	REAL	Order of the filter. Selects the first or second order filter algorithm. If invalid, the instruction sets the appropriate bit to true in Status and uses Order = 2.

Input Parameter	Data Type	Description
		Valid = 1 to 2 Default = 2
TimingMode	DINT	Selects timing execution mode. 0 = Periodic mode 1 = Oversample mode 2 = Real time sampling mode Valid = 0 to 2 Default = 0
RTSTimeStamp	DINT	Module time stamp value for real time sampling mode. Valid = 0 to 32,767ms Default = 0

Output Parameter	Data Type	Description
EnableOut	BOOL	Indicates if instruction is enabled. Cleared to false if Out overflows.
Out	REAL	The calculated output of the algorithm.
DeltaT	REAL	Elapsed time between updates. This is the elapsed time in seconds used by the control algorithm to calculate the process output.
Status	DINT	Status of the function block.
InstructFault (Status.0)	BOOL	The instruction detected one of the following execution errors. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.
WLeadInv (Status.1)	BOOL	WLead < minimum value or WLead > maximum value.
WLagInv (Status.2)	BOOL	WLag < minimum value or WLag > maximum value.
ZetaLeadInv (Status.3)	BOOL	Lead damping factor < minimum value or lead damping factor > maximum value.
ZetaLagInv (Status.4)	BOOL	Lag damping factor < minimum value or lag damping factor > maximum value.

Output Parameter	Data Type	Description
OrderInv (Status.5)	BOOL	Invalid Order value.
WLagRatioInv (Status.6)	BOOL	WLag:WLead ratio greater than maximum value.
TimingModeInv (Status.27)	BOOL	Invalid TimingMode value. For more information about timing modes, see Function Block Attributes.
RTSMissed (Status.28)	BOOL	Only used in real time sampling mode. Set when ABS (DeltaT - RTSTime) > 1 millisecond.
RTSTimeInv (Status.29)	BOOL	Invalid RTSTime value.
RTSTimeStampInv (Status.30)	BOOL	Invalid RTSTimeStamp value.
DeltaTInv (Status.31)	BOOL	Invalid DeltaT value.

Structured Text

Operand	Type	Format	Description
LDL2 tag	LEAD_LAG_SEC_ORDER	structure	LDL2 structure

See Structured Text Syntax for more information on the syntax of expressions within structured text.

Description

The LDL2 instruction filter is used in reference forcing and feedback forcing control methodologies. The LDL2 instruction is designed to execute in a task where the scan rate remains constant.

The LDL2 instruction uses these equations:

When:	The instruction uses this Laplace transfer function:
Order = 1	$H(s) = \frac{\frac{s}{\omega_{Lead}} + 1}{\frac{s}{\omega_{Lag}} + 1}$

When:	The instruction uses this Laplace transfer function:
Order = 2	$$H(s) = \frac{\frac{s^2}{\omega_{Lead}^2} + \frac{2 \times \xi_{Lead} \times s}{\omega_{Lead}} + 1}{\frac{s^2}{\omega_{Lag}^2} + \frac{2 \times \xi_{Lag} \times s}{\omega_{Lag}} + 1}$$ Normalize the filter such that $\omega_{Lead} = 1$ $$H(s) = \frac{s^2 + 2 \times \xi_{Lead} \times s + 1}{\frac{s^2}{\omega_{Lag}^2} + \frac{2 \times \xi_{Lag} \times s}{\omega_{Lag}} + 1}$$

with these parameter limits (where DeltaT is in seconds):

Parameter	Limitations
WLead first order LowLimit	$\frac{0.0000001}{DeltaT}$
WLead second order LowLimit	$\frac{0.00001}{DeltaT}$
HighLimit	$\frac{0.7\pi}{DeltaT}$
WLead:WLag ratio	If WLead > WLag, no limitations If WLag > WLead: • No minimum limitation for WLag:WLead • First order maximum for WLag:WLead = 40:1 and the instruction limits WLag to enforce this ratio • Second order maximum for WLag:WLead = 10:1 and the instruction limits WLag to enforce this ratio
ZetaLead second order only	LowLimit = 0.0 HighLimit = 4.0
ZetaLag second order only	LowLimit = 0.05 HighLimit = 4.0

Whenever the value computed for the output is invalid, NAN, or ± INF, the instruction sets Out = the invalid value. When the value computed for the output becomes valid, the instruction initializes the internal parameters and sets Out = In.

Affects Math Status Flags

No

Major/Minor Faults

None specific to this instruction. See Common Attributes for operand-related faults.

Execution

Function Block

Condition/State	Action Taken
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is true	EnableIn and EnableOut bits are set to true. The instruction executes.
Instruction first run	N/A
Instruction first scan	Recalculate coefficients.
Postscan	EnableIn and EnableOut bits are cleared to false.

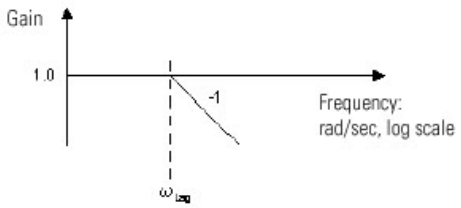
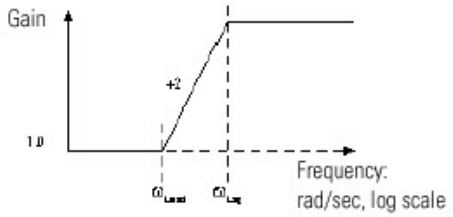
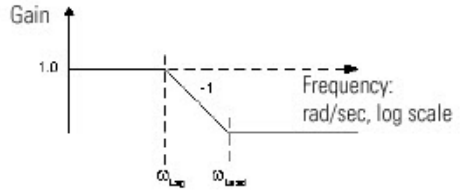
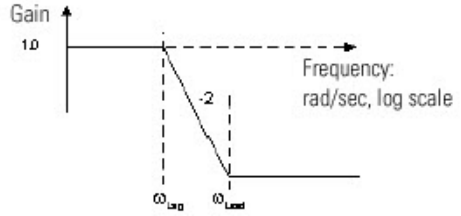
Structured Text

Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.
Normal Execution	See Tag.EnableIn is true in the Function Block table.
Postscan	See Postscan in the Function Block table.

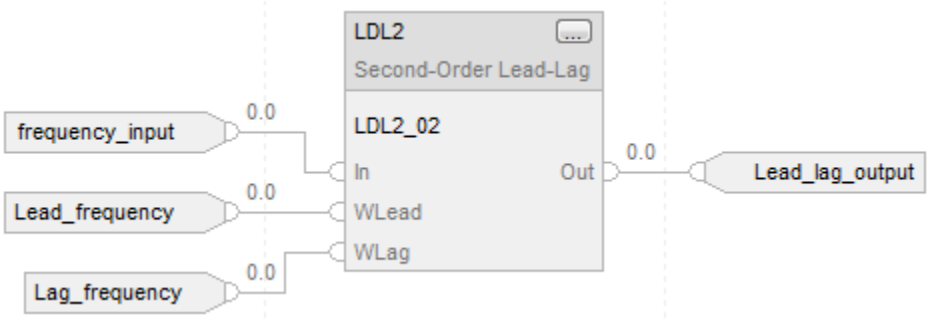
Example

The LDL2 instruction can attenuate between two frequencies or can amplify between two frequencies, depending on how you configure the instruction. Since the Lead and Lag frequencies can be set to values that are larger or smaller than each other, this instruction may behave as a Lead-Lag block, or, as a Lag-Lead block, depending on which frequency is configured first. Note that higher orders increase the execution time for the filter instruction.

This example is the minimal legal programming of the LDL2 function block and is only used to show the neutral text and generated code for this instruction. This is for internal purposes only and is not a testable case.

Filter	Graph
1st order lead-lag (wLead < wLag)	
2nd order lead-lag (wLead < wLag)	
1st order lead-lag (wLag < wLead)	
2nd order lead-lag (wLag < wLead)	

Function Block



Structured Text

```
LDL2_01.In := frequency_input;
LDL2_01.WLead :=
Lead_frequency;
LDL2_01.WLag := Lag_frequency;
LDL2(LDL2_01);
Lead_lag_output := LDL2_01.Out;
```

Related information

Function Block Attributes on page 546

Common Attributes on page

Structured Text Syntax on page 563

Select/Limit Instructions

The Select/Limit instructions include these instructions:

Available Instructions

Ladder Diagram

This instruction is not available in Ladder Diagram.

Function Block and Structured Text

ESEL on page 405	HLL on page 413	MUX on page 418	RLIM on page 421	SEL on page 425	SNEG on page 428	SSUM on page 430

If you want to	Use this instruction
Select one of as many as six inputs.	ESEL
Limit an analog input between two values.	HLL
Select one of eight inputs.	MUX
Limit the amount of change of a signal over time.	RLIM
Select one of two inputs.	SEL
Select between the input value and the negative of the input value.	SNEG
Select real inputs to be summed.	SSUM

Enhanced Select (ESEL)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, Compact GuardLogix 5380, CompactLogix 5380, ControlLogix 5580, GuardLogix 5580, and ControlLogix 5590 controllers.

The Enhanced Select (ESEL) instruction lets you select one of as many as six inputs. Selection options include:

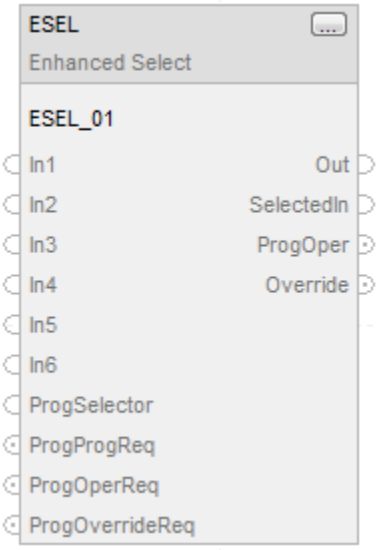
- Manual select (by operator or by program)
- High select
- Low select
- Median select
- Average (mean) select

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram logic.

Function Block



Structured Text

ESEL(ESEL_tag);

Operands

Function Block

Operand	Type	Format	Description
ESEL tag	SELECT_ENHANCED	Structure	ESEL structure

SELECT_ENHANCED Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If false, the instruction does not execute and outputs are not updated. Default is true.
In1	REAL	The first analog signal input to the instruction. Valid = any float Default = 0.0
In2	REAL	The second analog signal input to the instruction.

		Valid = any float Default = 0.0
In3	REAL	The third analog signal input to the instruction. Valid = any float Default = 0.0
In4	REAL	The fourth analog signal input to the instruction. Valid = any float Default = 0.0
In5	REAL	The fifth analog signal input to the instruction. Valid = any float Default = 0.0
In6	REAL	The sixth analog signal input to the instruction. Valid = any float Default = 0.0
In1Fault	BOOL	Bad health indicator for In1. If In1 is read from an analog input, then In1Fault is normally controlled by the fault status on the analog input. If all the InnFault inputs are true, the instruction sets the appropriate bit in Status, the control algorithm is not executed, and Out is not updated. Default = false
In2Fault	BOOL	Bad health indicator for In2. If In2 is read from an analog input, then In2Fault is normally controlled by the fault status on the analog input. If all the InnFault inputs are true, the instruction sets the appropriate bit in Status, the control algorithm is not executed, and Out is not updated. Default = false
In3Fault	BOOL	Bad health indicator for In3. If In3 is read from an analog input, then In3Fault is normally controlled by the fault status on the analog input. If all the InnFault inputs are true, the instruction sets the appropriate bit in Status, the control algorithm is not executed, and Out is not updated. Default = false
In4Fault	BOOL	Bad health indicator for In4. If In4 is read from an analog input, then In4Fault is normally controlled by the

		fault status on the analog input. If all the InnFault inputs are true, the instruction sets the appropriate bit in Status, the control algorithm is not executed, and Out is not updated. Default = false
In5Fault	BOOL	Bad health indicator for In5. If In5 is read from an analog input, then In5Fault is normally controlled by the fault status on the analog input. If all the InnFault inputs are true, the instruction sets the appropriate bit in Status, the control algorithm is not executed, and Out is not updated. Default = false
In6Fault	BOOL	Bad health indicator for In6. If In6 is read from an analog input, then In6Fault is normally controlled by the fault status on the analog input. If all the InnFault inputs are true, the instruction sets the appropriate bit in Status, the control algorithm is not executed, and Out is not updated. Default = false
InsUsed	DINT	Number of inputs used. This defines the number of inputs the instruction uses. The instruction considers only In1 through $In_{InsUsed}$ in high select, low select, median select, and average select modes. If this value is invalid, the instruction sets the appropriate bit in status. The instruction does not update Out if InsUsed is invalid and if the instruction is not in manual select mode and if Override is cleared. Valid =1 to 6 Default = 1
Selector Mode	DINT	Selector mode input. This value determines the action of the instruction. 0 = manual select 1 = High select 2 = Low select 3 = Median select 4 = Average select

		If this value is invalid, the instruction sets the appropriate bit in Status and does not update Out. Valid = 0 to 4 Default = 0
ProgSelector	DINT	Program selector input. When the selector mode is manual select and the instruction is in Program control, ProgSelector determines which input (In1-In6) to move into Out. If ProgSelector = 0, the instruction does not update Out. If ProgSelector is invalid, the instruction sets the appropriate bit in Status. If invalid and the instruction is in Program control, and the selector mode is manual select or Override is set, the instruction does not update. Out. Valid = 0 to 6 Default = 0
OperSelector	DINT	Operator selector input. When the selector mode is manual select and the instruction is in Operator control, OperSelector determines which input (In1-In6) to move into Out. If OperSelector = 0, the instruction does not update Out. If OperSelector is invalid, the instruction sets the appropriate bit in Status. If invalid and the instruction is in Operator control, and the selector mode is manual select or Override is set, the instruction does not update Out. Valid = 0 to 6 Default = 0
ProgProgReq	BOOL	Program program request. Set to true by the user program to request Program control. Ignored if ProgOperReq is true. Holding this true and ProgOperReq false locks the instruction into Program control. Default is false.
ProgOperReq	BOOL	Program operator request. Set to true by the user program to request Operator control. Holding this true locks the instruction into Operator control. Default is false.
ProgOverrideReq	BOOL	Program override request. Set to true by the user program to request

		the device to enter Override mode. In Override mode, the instruction will act as a manual select. Default is false.
OperProgReq	BOOL	Operator program request. Set to true by the operator interface to request Program control. The instruction clears this input to false. Default is false.
OperOperReq	BOOL	Operator operator request. Set to true by the operator interface to request Operator control. The instruction clears this input to false. Default is false.
ProgValueReset	BOOL	Reset program control values. When true, all the program request inputs are cleared to false on each execution of the instruction. Default is false.

Output Parameter	Data Type	Description
EnableOut	BOOL	Indicates if instruction is enabled. Cleared to false if Out overflows.
Out	REAL	The calculated output of the algorithm.
SelectedIn	DINT	Number of input selected. The instruction uses this value to display the number of the input currently being placed into the output. If the selector mode is average select, the instruction sets SelectedIn = 0.
ProgOper	BOOL	Program/Operator control indicator. Set to true when in Program control. Cleared to false when in Operatorcontrol.
Override	BOOL	Override mode. Set to true when the instruction is in Override mode.
Status	DINT	Status of the function block.
InstructFault (Status.0)	BOOL	The instruction detected one of the following execution errors. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.

InsFaulted (Status.1)	BOOL	InnFault inputs for all the used Inn inputs are true.
InsUsedInv (Status.2)	BOOL	Invalid InsUsed value.
SelectorModeInv (Status.3)	BOOL	Invalid SelectorMode value.
ProgSelectorInv (Status.4)	BOOL	Invalid ProgSelector value.
OperSelectorInv (Status.5)	BOOL	Invalid OperSelector value.

Structured Text

Operand	Type	Format	Description
ESEL tag	SELECT_ENHANCED	structure	ESEL structure

See Structured Text Syntax for more information on the syntax of expressions within structured text.

Description

The ESEL instruction operates as follows

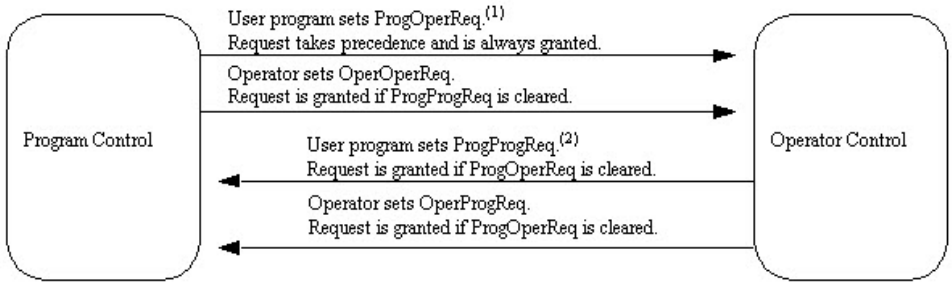
Condition	Action
SelectorMode = 0 (manual select) or Override is true, ProgOper is false and OperSelector is not equal to 0	Out = In[OperSelector] SelectedIn = OperSelector
SelectorMode = 0 (manual select) or Override is true, ProgOper is true and ProgSelector is not equal to 0	Out = In[ProgSelector] SelectedIn = ProgSelector
SelectorMode = 1 (high select) and Override is false	Out = maximum of In[InsUsed] SelectedIn = index to the maximum input value
SelectorMode = 2 (low select) and Override is false	Out = minimum of In[InsUsed] SelectedIn = index to the minimum input value
SelectorMode = 3 (median select) and Override is false	Out = median of In[InsUsed] SelectedIn = index to the median input value
SelectorMode = 4 (average select) and Override is false	Out = average of In[InsUsed] SelectedIn = 0

For SelectorMode 1 through 4, a bad health indication for any of the inputs causes that bad input to be disregarded in the selection. For example, if SelectorMode = 1 (high select) and if In6 had the highest value but had bad health, then the next highest input with good health is moved into the output.

For high or low select mode, if two inputs are equal and are high or low, the instruction outputs the first found input. For median select mode, the median value always represents a value selected from the available inputs. If more than one value could be the median, the instruction outputs the first found input.

Switch Between Program Control and Operator Control

The following diagram shows how the ESEL instruction changes between Program control and Operator control.



(1) You can lock the instruction in Operator control mode by leaving ProgOperReq true.

(2) You can lock the instruction in Program control mode by leaving ProgProgReq true while ProgOperReq is false.

Affects Math Status Flags

No

Major/Minor Faults

A minor fault will occur if the feature is enabled and overflow is detected (Fault Type: 4, Fault Code: 4).

Execution

Function Block

Condition/State	Action Taken
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is true	EnableIn and EnableOut bits are set to true. The instruction executes
Instruction first run	The instruction is set to Operator control.
Instruction first scan	N/A
Postscan	EnableIn and EnableOut bits are cleared to false.

Structured Text

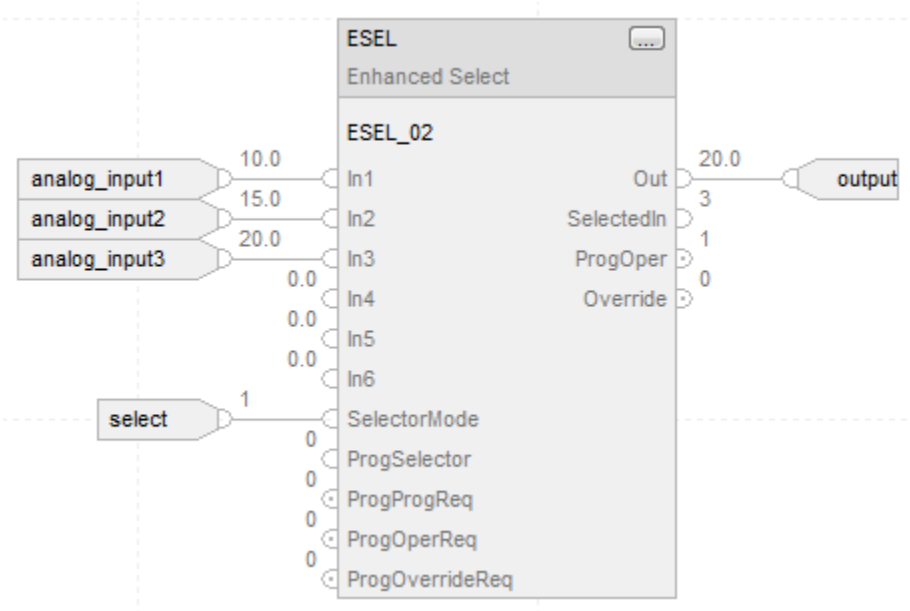
Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.
Normal Execution	See Tag.EnableIn is true in the Function Block table.

Postscan	See Postscan in the Function Block table.

Example

This ESEL instruction selects In1, In2, or In3, based on the SelectorMode. In this example, SelectorMode = 1, which means high select. The instruction determines which input value is the greatest and sets Out = greatest In.

Function Block



Structured Text

```
ESEL_01.In1 := analog_input1;
ESEL_01.In2 := analog_input2;
ESEL_01.In3 := analog_input3;
ESEL_01.SelectorMode := 1;
ESEL(ESEL_01);

selected_value := ESEL_01.Out;
```

High/Low Limit (HLL)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, Compact GuardLogix 5380, CompactLogix 5380, ControlLogix 5580, GuardLogix 5580, and ControlLogix 5590 controllers.

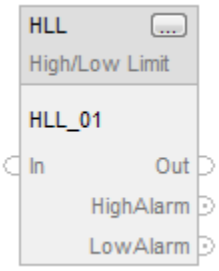
The High/Low Limit (HLL) instruction limits an analog input between two values. You can select high/low, high, or low limits.

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram logic.

Function Block



Structured Text

HLL(HLL_tag);

Operands

Function Block

Operand	Type	Format	Description
HLL tag	HL_LIMIT	structure	HLL structure

HL_LIMIT Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If false, the instruction does not execute and outputs are not updated. Default is true.
In	REAL	The analog signal input to the instruction. Valid = any float Default = 0.0
HighLimit	REAL	The high limit for the Input. If SelectLimit = 0 and HighLimit ≤ LowLimit, the instruction sets the appropriate bit in Status and sets Out = LowLimit. Valid = HighLimit > LowLimit Default = 0.0
LowLimit	REAL	The low limit for the Input. If SelectLimit = 0 and LowLimit ≥ HighLimit, the instruction sets the appropriate bit in Status and sets Out = LowLimit.

		Valid = LowLimit < HighLimit Default = 0.0
SelectLimit	DINT	Select limit input. This input has three settings: 0 = Use both limits 1= Use high limit 2 = Use low limit If SelectLimit is invalid, the instruction assumes SelectLimit = 0 and sets the appropriate bit in Status. Valid = 0 to 2 Default = 0

Output Parameter	Data Type	Description
EnableOut	BOOL	Indicates if instruction is enabled. Cleared to false if Out overflows.
Out	REAL	The calculated output of the algorithm.
HighAlarm	BOOL	The high alarm indicator. Set to true when In ≥ HighLimit. The HighAlarm is disabled when SelectLimit is set to 2.
LowAlarm	BOOL	The low alarm indicator. Set to true when In ≤ LowLimit. The LowAlarm is disabled when SelectLimit is set to 1.
Status	DINT	Status of the function block.
InstructFault (Status.0)	BOOL	The instruction detected one of the following execution errors. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.
LimitsInv (Status.1)	BOOL	HighLimit ≤ LowLimit.
SelectLimitInv (Status.2)	BOOL	The value of SelectLimit is not a 0, 1, or 2.

Structured Text

Operand	Type	Format	Description
HLL tag	HL_LIMIT	structure	HLL structure

See Structured Text Syntax for more information on the syntax of expressions within structured text.

Description

The HLL instruction determines the value of the Out using these rules:

Selection	Condition	Action
SelectLimit = 0 (use high and low limits)	In < HighLimit and In > LowLimit	Out = In
	In ≥ HighLimit	Out = HighLimit
	In ≤ LowLimit	Out = LowLimit
	HighLimit ≤ LowLimit	Out = LowLimit
SelectLimit = 1 (use high limit only)	In < HighLimit	Out = In
	In≥HighLimit	Out = HighLimit
SelectLimit = 2 (use low limit only)	In > LowLimit	Out = In
	In ≤ LowLimit	Out = LowLimit

Affects Math Status Flags

No

Major/Minor Faults

A minor fault will occur if the feature is enabled and overflow is detected (Fault Type: 4, Fault Code: 4).

Execution

Function Block

Condition/State	Action Taken
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is true	EnableIn and EnableOut bits are set to true.

	The instruction executes.
Instruction first run	N/A
Instruction first scan	N/A
Postscan	EnableIn and EnableOut bits are cleared to false.

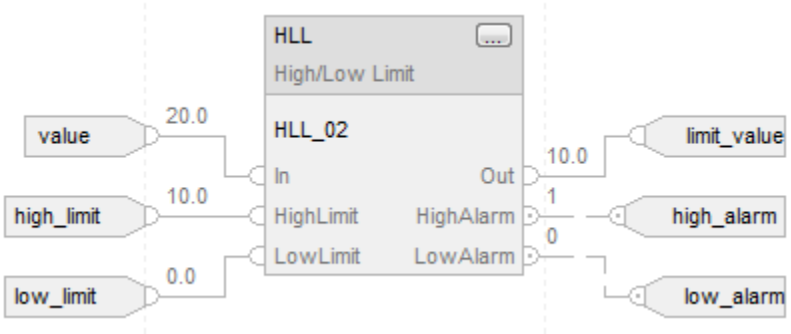
Structured Text

Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.
Normal Execution	See Tag.EnableIn is true in the Function Block table.
Postscan	See Postscan in the Function Block table.

Example

This HLL instruction limits In between two values and sets HighAlarm or LowAlarm, if needed when In is outside the limits. The instruction sets Out = limited value of In.

Function Block



Structured Text

```
HLL_01.In := value;
HLL_01.HighLimit := high_limit;
HLL_01.LowLimit := low_limit;
HLL(HLL_01);
limited_value := HLL_01.Out;
high_alarm := HLL_01.HighAlarm;
low_alarm := HLL_01.LowAlarm;
```

Multiplexer (MUX)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, Compact GuardLogix 5380, CompactLogix 5380, ControlLogix 5580, GuardLogix 5580, and ControlLogix 5590 controllers.

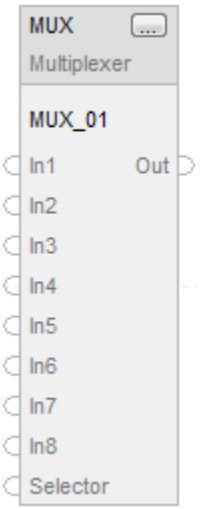
The Multiplexer (MUX) instruction selects one of eight inputs based on the selector input.

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram.

Function Block



Structured Text

This instruction is not available in ladder diagram.

Operands

Function Block

Operand	Type	Format	Description
Block tag	MULTIPLEXER	Structure	MUX structure

MUX Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.

In1	REAL	The first analog signal input to the instruction. Valid = any float Default = 0.0
In2	REAL	The second analog signal input to the instruction. Valid = any float Default = 0.0
In3	REAL	The third analog signal input to the instruction. Valid = any float Default = 0.0
In4	REAL	The fourth analog signal input to the instruction. Valid = any float Default = 0.0
In5	REAL	The fifth analog signal input to the instruction. Valid = any float Default = 0.0
In6	REAL	The sixth analog signal input to the instruction. Valid = any float Default = 0.0
In7	REAL	The seventh analog signal input to the instruction. Valid = any float Default = 0.0
In8	REAL	The eighth analog signal input to the instruction. Valid = any float Default = 0.0
Selector	DINT	The selector input to the instruction. This input determines which of the inputs (1-8) is moved into Out. If this value is invalid (which includes 0), the instruction sets the appropriate bit in Status and holds Out at its current value. Valid = 1 to 8 Default = 0

Output Parameter	Data Type	Description
EnableOut	BOOL	Indicates if the instruction is enabled. Cleared on overflow.

Out	REAL	The selected output of the algorithm. Math status flags are set for this output.
Status	DINT	Status of the function block.
InstructFault (Status.0)	BOOL	The instruction detected one of the following execution errors. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.
SelectorInv (Status.1)	BOOL	Invalid Selector value.

Description

Based on the Selector value, the MUX instruction sets Out equal to one of eight inputs.

Affects Math Status Flags

No

Major/Minor Faults

A minor fault will occur if the feature is enabled and overflow is detected (Fault Type: 4, Fault Code: 4).

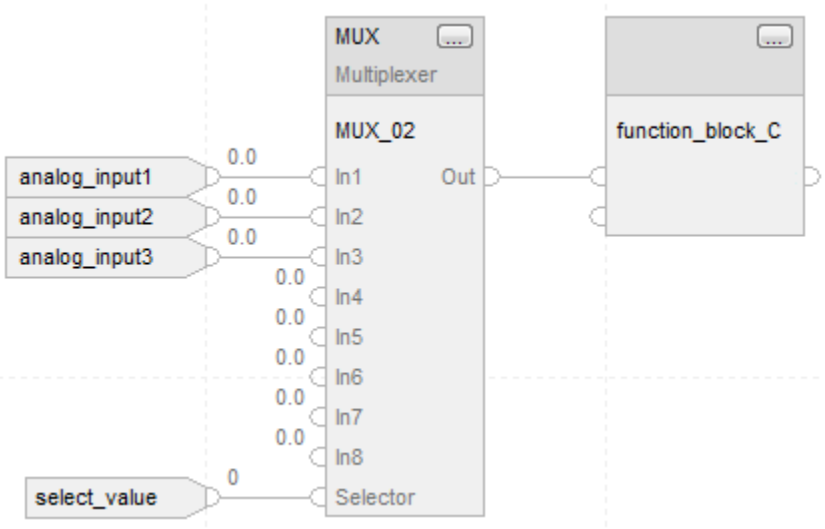
Execution

Function Block

Condition	Action
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is true	EnableIn and EnableOut bits are set to true. The instruction executes.
Instruction first run	N/A
Instruction first scan	Set internal value of Out to zero.
Postscan	EnableIn and EnableOut bits are cleared to false.

Example

Function Block



This MUX instruction selects In1, In2, or In3, In4, In5, In6, In7, or In8 based on the Selector. The instruction sets Out = In_n, which becomes an input parameter for function_block_C. For example, if select_value = 2, the instruction sets Out = analog_input2.

Rate Limiter (RLIM)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, Compact GuardLogix 5380, CompactLogix 5380, ControlLogix 5580, GuardLogix 5580, and ControlLogix 5590 controllers.

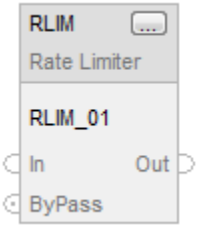
The Rate Limiter (RLIM) instruction limits the amount of change of a signal over time.

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram logic.

Function Block



Structured Text

RLIM(RLIM_tag);

Operands

Function Block

Operand	Type	Format	Description
RLIM tag	RATE_LIMITER	structure	RLIM structure

RATE_LIMITER Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If false, the instruction does not execute and outputs are not updated. Default is true.
In	REAL	The analog signal input to the instruction. Valid = any float Default = 0.0
IncRate	REAL	Maximum output increment rate in per-second units. If invalid, the instruction sets IncRate = 0.0 and sets the appropriate bit in Status. Valid = any float ≥ 0.0 Default = 0.0
DecRate	REAL	Maximum output decrement rate in per-second units. If invalid, the instruction sets DecRate = 0.0 and sets the appropriate bit in Status. Valid = any float ≥ 0.0 Default = 0.0
ByPass	BOOL	Request to bypass the algorithm. When true, Out = In. Default is false.
TimingMode	DINT	Selects timing execution mode. 0 = Period mode 1 = oversample mode 2 = real time sampling mode Valid = 0 to 2 Default = 0
OversampleDT	REAL	Execution time for oversample mode. Valid = 0 to 4194.303 seconds Default = 0

RTSTime	DINT	Module update period for real time sampling mode Valid = 1 to 32,767ms Default = 1
RTSTimeStamp	DINT	Module time stamp value for real time sampling mode. Valid = 0 to 32,767ms Default = 0

Output Parameter	Data Type	Description
EnableOut	BOOL	Indicates if instruction is enabled. Cleared to false if Out overflows.
Out	REAL	The calculated output of the algorithm.
DeltaT	REAL	Elapsed time between updates. This is the elapsed time in seconds used by the control algorithm to calculate the process output.
Status	DINT	Status of the function block.
InstructFault (Status.0)	BOOL	The instruction detected one of the following execution errors. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.
IncRateInv (Status.1)	BOOL	IncRate < 0. The instruction uses 0.
DecRateInv (Status.2)	BOOL	DecRate < 0. The instruction uses 0.
TimingModeInv (Status.27)	BOOL	Invalid TimingMode value. For more information about timing modes, see Function Block Attributes.
RTSMissed (Status.28)	BOOL	Only used in real time sampling mode. Set to true when ABS(DeltaT – RTSTime) > 1 millisecond.
RTSTimeInv (Status.29)	BOOL	Invalid RTSTime value.
RTSTimeStampInv (Status.30)	BOOL	Invalid RTSTimeStamp value.
DeltaTInv (Status.31)	BOOL	Set to true in oversample mode if either DeltaT <= 0 or DeltaT > 4194.303.

Structured Text

Operand	Type	Format	Description
RLIM tag	RATE_LIMITER	structure	RLIM structure

See Structured Text Syntax for more information on the syntax of expressions within structured text.

Description

The RLIM instruction provides separate increment and decrement rates in per-second units. The ByPass input lets you stop rate limiting and pass the signal directly to the output.

Condition	Action
ByPass is true	$Out_n = In_n$ $Out_{n-1} = In_n$
ByPass is false and DeltaT > 0	$Slope = \frac{In_n - Out_{n-1}}{DeltaT}$ If Slope ≤ –DecRate then YSlope = –DecRate If –DecRate ≤ Slope ≤ IncRate then YSlope = Slope If IncRate ≤ Slope then YSlope = IncRate $Out_n = Out_{n-1}$ + DeltaT x YSlope $Out_{n-1} = Out_n$ where DeltaT is in seconds

Affects Math Status Flags

No

Major/Minor Faults

A minor fault will occur if the feature is enabled and overflow is detected (Fault Type: 4, Fault Code: 4).

Execution

Function Block

Condition/State	Action Taken
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is true	EnableIn and EnableOut bits are set to true The instruction executes.
Instruction first run	N/A
Instruction first scan	Initialize Out with the value of In.

Postscan	EnableIn and EnableOut bits are cleared to false.

Structured Text

Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.
Normal Execution	See Tag.EnableIn is true in the Function Block table.
Postscan	See Postscan in the Function Block table.

Example

Function Block

The RLIM instruction limits In by IncRate. If analog_input1 changes at a rate greater than the IncRate value, the instruction limits In. The instruction sets Out = rate limited value of In.

Structured Text

```
RLIM_01.In := analog_input1;
RLIM_01.IncRate := value;
RLIM(RLIM_01);

rate_limited := RLIM_01.Out;
```

Select (SEL)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, Compact GuardLogix 5380, CompactLogix 5380, ControlLogix 5580, GuardLogix 5580, and ControlLogix 5590 controllers.

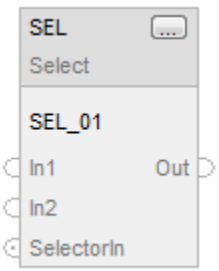
The Select (SEL) instruction uses a digital input to select one of two inputs.

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram logic.

Function Block



Structured Text

This instruction is not available in structured text.

Operands

Function Block

Operand	Type	Format	Description
SEL tag	SELECT	structure	SEL structure

SELECT Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
In1	REAL	The first analog signal input to the instruction. Valid = any float Default = 0.0
In2	REAL	The second analog signal input to the instruction. Valid = any float Default = 0.0
SelectorIn	BOOL	The input that selects between In1 and In2. Default is cleared.

Output Parameter	Data Type	Description
EnableOut	BOOL	Indicates if instruction is enabled. Cleared on overflow.
Out	REAL	The calculated output of the algorithm.

Description

The SEL instruction operates as follows:

Condition	Action
SelectorIn is set	Out = In2
SelectorIn is cleared	Out = In1

Affects Math Status Flags

No

Major/Minor Faults

A minor fault will occur if the feature is enabled and overflow is detected (Fault Type: 4, Fault Code: 4).

Execution

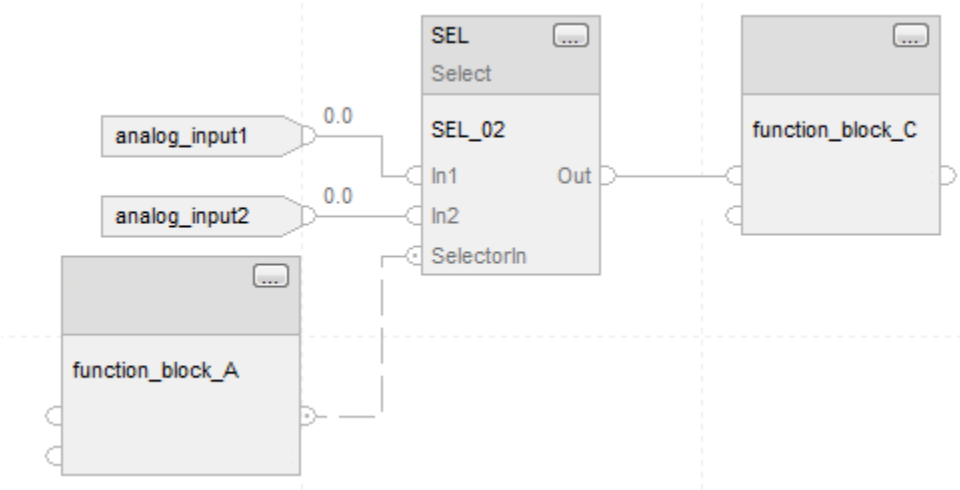
Function Block

Condition/State	Action Taken
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is true	EnableIn and EnableOut bits are set to true. The instruction executes.
Instruction first run	N/A
Instruction first scan	Out n-1 is set to 0.
Postscan	.EnableIn and .EnableOut bits are cleared to false.

Example

The SEL instruction selects In1 or In2 based on SelectorIn. If SelectorIn is set, the instruction sets Out = In2. If SelectorIn is cleared, the instruction sets Out = In1. Out becomes an input parameter for function_block_C.

Function Block



Selected Negate (SNEG)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, Compact GuardLogix 5380, CompactLogix 5380, ControlLogix 5580, GuardLogix 5580, and ControlLogix 5590 controllers.

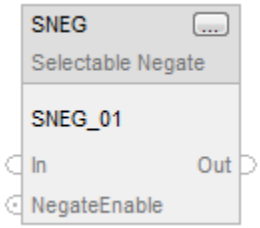
The Selected Negate (SNEG) instruction uses a digital input to select between the input value and the negative of the input value.

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram logic.

Function Block



Structured Text

SNEG(SNEG_tag);

Operands

Function Block

Operand	Type	Format	Description
SNEG tag	SELECTABLE_NEGATE	Structure	SNEG structure

SNEG Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If false, the instruction does not execute and outputs are not updated. Default is true. Structured Text: No effect. The instruction executes.
In	REAL	The analog signal input to the instruction. Valid = any float Default = 0.0

NegateEnable	BOOL	Negate enable. When NegateEnable is true, the instruction sets Out to the negative value of In. Default is true.

Output Parameter	Data Type	Description
EnableOut	BOOL	Indicates if instruction is enabled. Cleared to false if Out overflows.

Structured Text

Operand	Type	Format	Description
SNEG tag	SELECTABLE_NEGATE	Structure	SNEG structure

See Structured Text Syntax for more information on the syntax of expressions within structured text.

Description

The SNEG instruction operates as follows:

Condition	Action
NegateEnable is true	Out = - In
NegateEnable is false	Out = In

Affects Math Status Flags

No

Major/Minor faults

None specific to this instruction. See Common Attributes on page for operand-related faults.

Execution

Function Block

Condition/State	Action Taken
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is true	EnableIn and EnableOut bits are set to true. The instruction executes.

Instruction first run	N/A
Instruction first scan	N/A
Postscan	EnableIn and EnableOut bits are cleared to false.

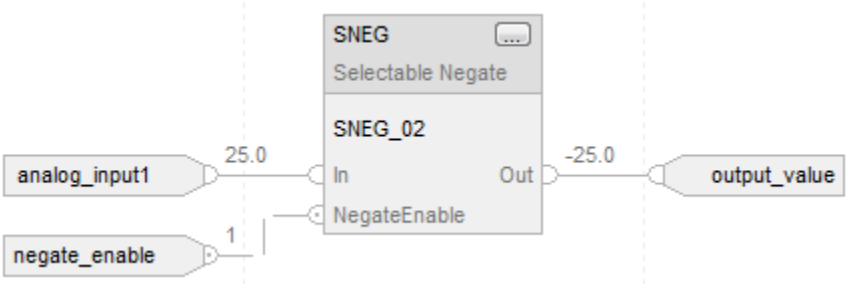
Structured Text

Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.
Normal Execution	See Tag.EnableIn is true in the Function Block table.
Postscan	See Postscan in the Function Block table.

Example

The negate_enable input determines whether to negate In or not. The instruction sets Out = In if NegateEnable is false. The instruction sets Out = -In if NegateEnable is true.

Function Block



Structured Text

```
SNEG_01.In := analog_input1;
SNEG_01.NegateEnable := negate_enable;
SNEG(SNEG_01);

output_value := SNEG_01.Out;
```

Selected Summer (SSUM)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, Compact GuardLogix 5380, CompactLogix 5380, ControlLogix 5580, GuardLogix 5580, and ControlLogix 5590 controllers.

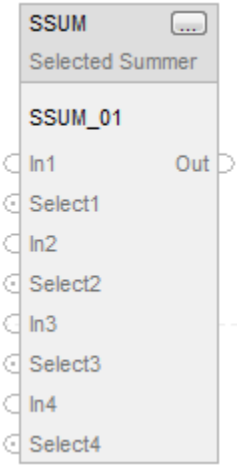
The Selected Summer (SSUM) instruction uses Boolean inputs to select real inputs to be algebraically summed.

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram logic.

Function Block



Structured Text

SSUM(SSUM_tag);

Operands

Function Block

Operand	Type	Format	Description
SSUM tag	SELECTED_SUMMER	Structure	SSUM structure

SELECTABLE_SUMMER Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If false, the instruction does not execute and outputs are not updated. Default is true.
In1	REAL	The first input to be summed. Valid = any float Default = 0.0
Gain1	REAL	Gain for the first input. Valid = any float Default = 1.0
Select1	BOOL	Selector signal for the first input.

		Default is false.
In2	REAL	The second input to be summed. Valid = any float Default = 0.0
Gain2	REAL	Gain for the second input. Valid = any float Default = 1.0
Select2	BOOL	Selector signal for the second input. Default is false.
In3	REAL	The third input to be summed. Valid = any float Default = 0.0
Gain3	REAL	Gain for the third input. Valid = any float Default = 1.0
Select3	BOOL	Selector signal for the third input. Default is false.
In4	REAL	The fourth input to be summed. Valid = any float Default = 0.0
Gain4	REAL	Gain for the fourth input. Valid = any float Default = 1.0
Select4	BOOL	Selector signal for the fourth input. Default is false.
In5	REAL	The fifth input to be summed. Valid = any float Default = 0.0
Gain5	REAL	Gain for the fifth input. Valid = any float Default = 1.0
Select5	BOOL	Selector signal for the fifth input. Default is false.
In6	REAL	The sixth input to be summed. Valid = any float Default = 0.0
Gain6	REAL	Gain for the sixth input. Valid = any float Default = 1.0
Select6	BOOL	Selector signal for the sixth input. Default is false.
In7	REAL	The seventh input to be summed. Valid = any float

		Default = 0.0
Gain7	REAL	Gain for the seventh input. Valid = any float Default = 1.0
Select7	BOOL	Selector signal for the seventh input. Default is false.
In8	REAL	The eighth input to be summed. Valid = any float Default = 0.0
Gain8	REAL	Gain for the eighth input. Valid = any float Default = 1.0
Select8	BOOL	Selector signal for the eighth input. Default is false.
Bias	REAL	Bias signal input. The instruction adds the Bias to the sum of the inputs. Valid = any float Default = 0.0

Output Parameter	Data Type	Description
EnableOut	BOOL	Indicates if instruction is enabled. Cleared to false if Out overflows.
Out	REAL	The calculated output of the algorithm.

Structured Text

Operand	Type	Format	Description
SSUM tag	SELECTED_SUMMER	Structure	SSUM structure

See Structured Text Syntax for more information on the syntax of expressions within structured text.

Description

The SSUM instruction operates as follows:

Condition	Action
No In is selected	Out = Bias
One or more In are selected	For all n where Selectn is true

	$Out = \sum (In_n \times Gain_n) + Bias$

Affects Math Status Flags

No

Major/Minor Faults

A minor fault will occur if the feature is enabled and overflow is detected (Fault Type: 4, Fault Code: 4).

Execution

Function Block

Condition/State	Action Taken
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is true	EnableIn and EnableOut bits are set to true. The instruction executes.
Instruction first run	N/A
Instruction first scan	N/A
Postscan	EnableIn and EnableOut bits are cleared to false.

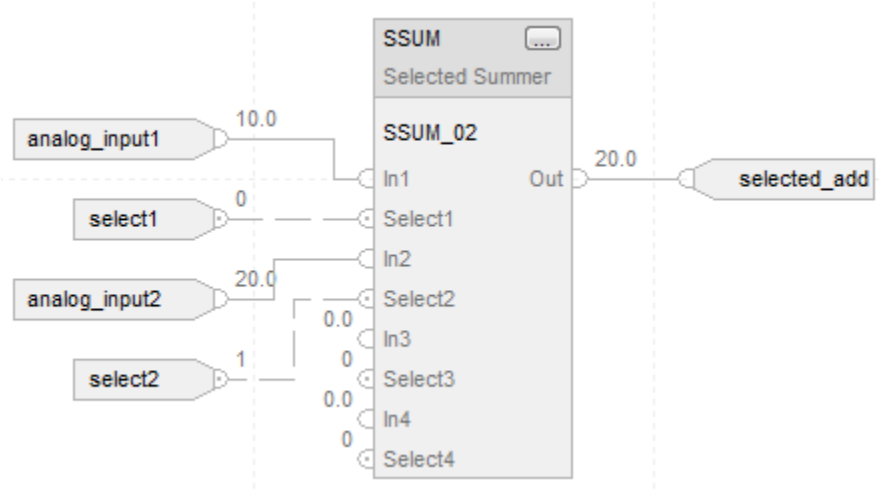
Structured Text

Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.
Normal Execution	See Tag.EnableIn is true in the Function Block table.
Postscan	See Postscan in the Function Block table.

Example

The values of select1 and select 2 determine whether to select analog_input1 and analog_input2, respectively. The instruction then adds the selected inputs and places the result in Out.

Function Block



Structured Text

```
SSUM_01.In1 := analog_input1;
SSUM_01.Select1 := select1;
SSUM_01.In2 := analog_input2;
SSUM_01.Select2 := select2;
SSUM(SSUM_01);

selected_add := SSUM_01.Out;
```

Statistical Instructions

The Statistical instructions include these instructions:

Available Instructions

Ladder Diagram

Not available

Function Block and Structured Text

MAVE on page 436	MAXC on page 443	MINC on page 446	MSTD on page 449

If you want to	Use this instruction
Calculate a time average value.	MAVE
Find the maximum signal in time.	MAXC
Find the minimum signal in time.	MINC
Calculate a moving standard deviation.	MSTD

Moving Average (MAVE)

This information applies to the CompactLogix 5370, ControlLogix 5570, CompactGuardLogix 5370, GuardLogix 5570, CompactGuardLogix 5380, CompactLogix 5380, ControlLogix 5580, GuardLogix 5580, and ControlLogix 5590 controllers.

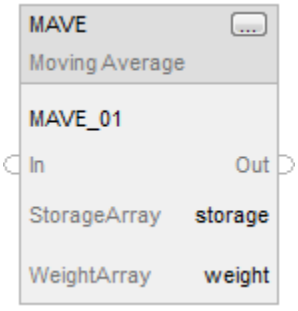
The Moving Average (MAVE) instruction calculates a time average value for the In signal. This instruction optionally supports user-specified weights.

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram logic.

Function Block



Structured Text

MAVE(MAVE_tag,storage,weight);

Operands

Function Block

Operand	Type	Format	Description
MAVE tag	MOVING_AVERAGE	structure	MAVE structure
storage	REAL	array	Holds the moving average samples. This array must be at least as large as NumberOfSamples.
weight	REAL	array	(optional) Used for weighted averages. This array must be at least as large as NumberOfSamples. Element [0] is used for the newest sample; element [n] is used for the oldest sample.

MOVING_AVERAGE Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
In	REAL	The analog signal input to the instruction. Valid = any float Default = 0.0
InFault	BOOL	Bad health indicator for the input. If In is read from an analog input, then InFault is normally controlled by fault status on the analog input. When set, InFault indicates the input signal has an error, the instruction sets the appropriate bit in Status, and the instruction holds Out at its current value. When InFault transitions from set to cleared, the instruction initializes the averaging algorithm and continues executing. Default is cleared.

Initialize	BOOL	Initialize input to the instruction. When set, the instruction holds Out = In, except when InFault is set, in which case, the instruction holds Out at its current value. When Initialize transitions from set to cleared, the instruction initializes the averaging algorithm and continues executing. Default is cleared.
SampleEnable	BOOL	Enable for taking a sample of In. When set, the instruction enters the value of In into the storage array and calculates a new Out value. When SampleEnable is cleared and Initialize is cleared, the instruction holds Out at its current value. Default is set.
NumberOfSamples	DINT	The number of samples to be used in the calculation. If this value is invalid, the instruction sets the appropriate bit in Status and holds Out at its current value. When NumberOfSamples becomes valid again, the instruction initializes the averaging algorithm and continues executing. Valid = 1 to (minimum size of StorageArray or WeightArray, if used) Default = 1
UseWeights	BOOL	Averaging scheme input to the instruction. When set, the instruction uses the weighted method to calculate the Out. When cleared, the instruction uses the uniform method to calculate Out. Default is cleared.

Output Parameter	Data Type	Description
EnableOut	BOOL	Indicates if instruction is enabled. Cleared to false if Out overflows.
Out	REAL	The calculated output of the algorithm.
Status	DINT	Status of the function block.
InstructFault (Status.0)	BOOL	The instruction detected one of the following execution errors. This is not a minor or major controller error.

		Check the remaining status bits to determine what occurred.
InFaulted (Status.1)	BOOL	In health is bad (InFault is set).
NumberOfSampInv (Status.2)	BOOL	NumberOfSamples invalid or not compatible with array size.

Structured Text

Operand	Type	Format	Description
MAVE tag	MOVING_AVERAGE	structure	MAVE structure
storage	REAL	array	Holds the moving average samples. This array must be at least as large as NumberOfSamples.
weight	REAL	array	(optional) Used for weighted averages. This array must be at least as large as NumberOfSamples. Element [0] is used for the newest sample; element [n] is used for the oldest sample.

MOVING_AVERAGE Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
In	REAL	The analog signal input to the instruction. Valid = any float Default = 0.0
InFault	BOOL	Bad health indicator for the input. If In is read from an analog input, then InFault is normally controlled by fault status on the analog input. When set, InFault indicates the input signal has an error, the instruction sets the appropriate bit in Status, and the instruction holds Out at its current value. When InFault transitions from set to cleared, the instruction

		initializes the averaging algorithm and continues executing. Default is cleared.
Initialize	BOOL	Initialize input to the instruction. When set, the instruction holds Out = In, except when InFault is set, in which case, the instruction holds Out at its current value. When Initialize transitions from set to cleared, the instruction initializes the averaging algorithm and continues executing. Default is cleared.
SampleEnable	BOOL	Enable for taking a sample of In. When set, the instruction enters the value of In into the storage array and calculates a new Out value. When SampleEnable is cleared and Initialize is cleared, the instruction holds Out at its current value. Default is set.
NumberOfSamples	DINT	The number of samples to be used in the calculation. If this value is invalid, the instruction sets the appropriate bit in Status and holds Out at its current value. When NumberOfSamples becomes valid again, the instruction initializes the averaging algorithm and continues executing. Valid = 1 to (minimum size of StorageArray or WeightArray, if used) Default = 1
UseWeights	BOOL	Averaging scheme input to the instruction. When set, the instruction uses the weighted method to calculate the Out. When cleared, the instruction uses the uniform method to calculate Out. Default is cleared.

Output Parameter	Data Type	Description
EnableOut	BOOL	Indicates if instruction is enabled. Cleared to false if Out overflows.
Out	REAL	The calculated output of the algorithm.
Status	DINT	Status of the function block.

InstructFault (Status.0)	BOOL	The instruction detected one of the following execution errors. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.
InFaulted (Status.1)	BOOL	In health is bad (InFault is set).
NumberOfSampInv (Status.2)	BOOL	NumberOfSamples invalid or not compatible with array size.

See Structured Text Syntax on page for more information on the syntax of expressions within structured text.

Description

The MAVE instruction calculates a weighted or non-weighted moving average of the input signal. The NumberOfSamples specifies the length of the moving average span. At every scan of the block when Sample Enable is set, the instruction moves the value of In into the storage array and discards the oldest value. Each In_n has a user-configured $Weight_n$, which is used if UseWeights is set.

Condition	Action
Weighted averaging method UseWeights is set.	$Out = \sum_{n=1}^{NumberOfSamples} Weight_n \times In_n$
Uniform averaging method UseWeights is cleared.	$Sum = \sum_{n=1}^{NumberOfSamples} In_n$ $Out = \frac{Sum}{NumberOfSamples}$

The instruction will not place an invalid In value (NAN or ± INF) into the storage array. When In is invalid, the instruction sets Out = In and logs an overflow minor fault, if this reporting is enabled. When In becomes valid, the instruction initializes the averaging algorithm and continues executing.

You can make runtime changes to the NumberOfSamples parameter. If you increase the number, the instruction incrementally averages new data from the current sample size to the new sample size. If you decrease the number, the instruction recalculates the average from the beginning of the sample array to the new NumberOfSamples value.

Initializing the Averaging Algorithm

Certain conditions, such as instruction first scan and instruction first run, require the instruction to initialize the moving average algorithm. When this occurs, the instruction

considers the sample StorageArray empty and incrementally averages samples from 1 to the NumberOfSamples value. For example:

NumberOfSamples = 3, UseWeights is set

Scan 1: Out = $In_n * Weight_1$

Scan 2: Out = $(In_n * Weight_1) + (In_{n-1} * Weight_2)$

Scan 3: Out = $(In_n * Weight_1) + (In_{n-1} * Weight_2) + (In_{n-2} * Weight_3)$

NumberOfSamples = 3, UseWeights is cleared

Scan 1: Out = $In_n / 1$

Scan 2: Out = $(In_n + In_{n-1}) / 2$

Scan 3: Out = $(In_n + In_{n-1} + In_{n-2}) / NumberOfSamples$

Affects Math Status Flags

No

Major/Minor Faults

None specific to this instruction. See Common Attributes on page 591 for operand-related faults.

Execution

Function Block

Condition/State	Action Taken
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false. Request re-initialization of Storage array the next time the instruction executes.
Tag.EnableIn is true	EnableIn and EnableOut bits are set to true. The instruction executes.
Instruction first run	N/A
Instruction first scan	Initialize Out to zero.
Postscan	EnableIn and EnableOut bits are cleared to false.

Structured Text

In Structured Text, EnableIn is always True during a normal scan. If the instruction is in the control path activated by the logic, it executes.

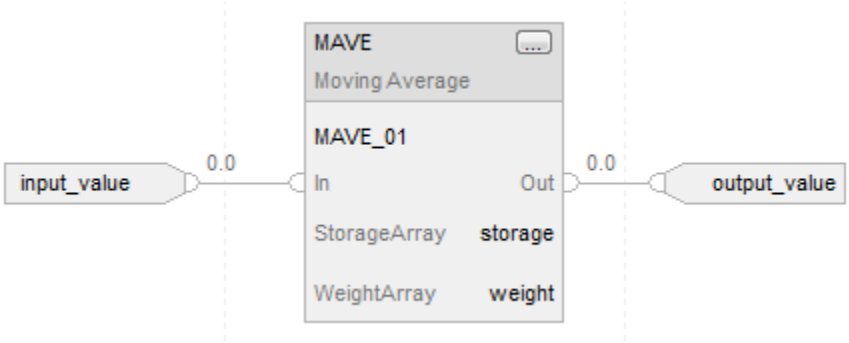
Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.

Normal Execution	See Tag.EnableIn is true in the Function Block table.
Postscan	See Postscan in the Function Block table.

Example

Each scan, the instruction places input_value in array storage. The instruction calculates the average of the values in array storage, optionally using the weight values in array weight, and places the result in Out.

Function Block



Structured Text

```
MAVE_01.In := input_value;
MAVE(MAVE_01,storage,weight);
output_value := MAVE_01.Out;
```

Maximum Capture (MAXC)

This information applies to the CompactLogix 5370, ControlLogix 5570, CompactGuardLogix 5370, GuardLogix 5570, CompactGuardLogix 5380, CompactLogix 5380, ControlLogix 5580, GuardLogix 5580, and ControlLogix 5590 controllers.

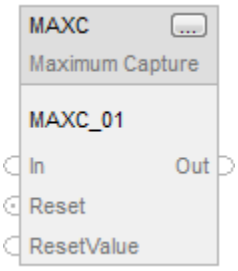
The Maximum Capture (MAXC) instruction retains the maximum value of the input over time and allows the user to re-establish a maximum as needed.

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram logic.

Function Block



Structured Text

MAXC(MAXC_tag);

Operands

Function Block

Operand	Type	Format	Description
MAXC tag	MAXIMUM_CAPTURE	Structure	MAXC structure

MAXIMUM_CAPTURE Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
In	REAL	The analog signal input to the instruction. Valid = any float Default = 0.0
Reset	BOOL	Request to reset control algorithm. The instruction sets Out = ResetValue as long as Reset is set. Default is cleared.
ResetValue	REAL	The reset value for the instruction. The instruction sets Out = ResetValue as long as Reset is set. Valid = any float Default = 0.0

Output Parameter	Data Type	Description
EnableOut	BOOL	Enable output.

Out	REAL	The calculated output of the algorithm.

Structured Text

Operand	Type	Format	Description
MAXC tag	MAXIMUM_CAPTURE	Structure	MAXC structure

See Structured Text Syntax on page for more information on the syntax of expressions within structured text.

Description

The MAXC instruction executes this algorithm:

Condition	Action
Reset is set	LastMaximum = Reset value Out = LastMaximum
Reset is cleared	If In > LastMaximum then update LastMaximum. Out = LastMaximum.

Affects Math Status Flags

No

Major/Minor Faults

None specific to this instruction. See Common Attributes on page 591 for operand-related faults.

Execution

Function Block

Condition/State	Action Taken
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is true	EnableIn and EnableOut bits are set to true. The instruction executes.
Instruction first run	N/A
Instruction first scan	Set request to initialize the Maximum value with the current input.
Postscan	EnableIn and EnableOut bits are cleared to false.

Structured Text

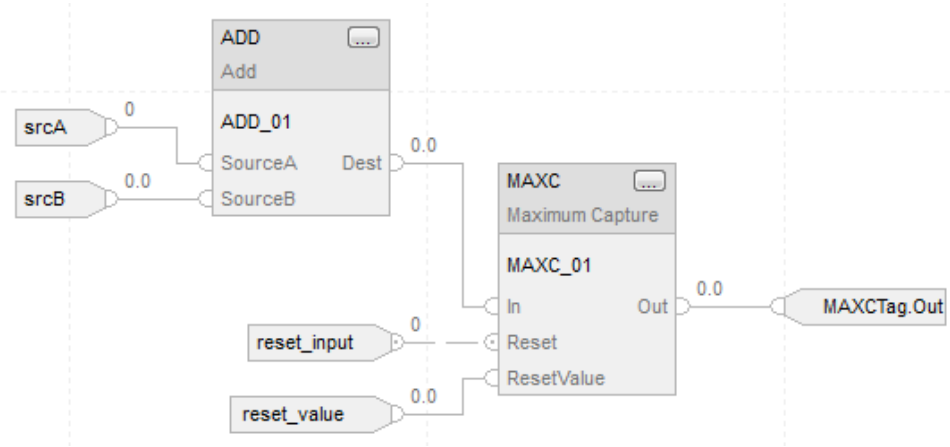
In Structured Text, EnableIn is always True during a normal scan. If the instruction is in the control path activated by the logic, it executes.

Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.
Normal Execution	See Tag.EnableIn is true in the Function Block table.
Postscan	See Postscan in the Function Block table.

Examples

If Reset is set, the instruction sets Out=ResetValue. If Reset is cleared, the instruction sets Out=In when In> LastMaximum. Otherwise, the instruction sets Out= LastMaximum.

Function Block



Structured Text

MAXCTag.In := input_value;

MAXCTag.Reset := reset_input;

MAXCTag.ResetValue := reset_value;

MAXC(MAXCTag);

result := MAXCTag.Out;

Minimum Capture (MINC)

This information applies to the CompactLogix 5370, ControlLogix 5570, CompactGuardLogix 5370, GuardLogix 5570, CompactGuardLogix 5380, CompactLogix 5380, ControlLogix 5580, GuardLogix 5580, and ControlLogix 5590 controllers.

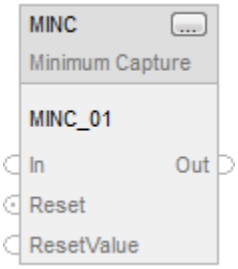
The Minimum Capture (MINC) instruction retains the minimum value of the input over time and allows the user to re-establish a minimum as needed.

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram.

Function Block



Structured Text

MINC(MINC_tag);

Operands

Function Block

Operand	Type	Format	Description
MINC tag	MINIMUM_CAPTURE	structure	MINC structure

MINIMUM_CAPTURE Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
In	REAL	The analog signal input to the instruction. Valid = any float Default = 0.0
Reset	BOOL	Request to reset control algorithm. The instruction sets Out = ResetValue as long as Reset is set. Default is cleared.
ResetValue	REAL	The reset value for the instruction. The instruction sets Out = ResetValue as long as Reset is set. Valid = any float Default = 0.0

Output Parameter	Data Type	Description
EnableOut	BOOL	Enable output.
Out	REAL	The calculated output of the algorithm.

Structured Text

Operand	Type	Format	Description
MINC tag	MINIMUM _CAPTURE	structure	MINC structure

See Structured Text Syntax on page for more information on the syntax of expressions within structured text.

Description

The MINC instruction executes this algorithm:

Condition	Action
Reset is set	LastMinimum = ResetValue Out = ResetValue
Reset is cleared	If In < LastMinimum then update LastMinimum. Out = LastMinimum.

Affects Math Status Flags

No

Major/Minor Faults

None specific to this instruction. See Common Attributes on page 591 for operand-related faults.

Execution

Function Block

Condition/State	Action Taken
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is true	EnableIn and EnableOut bits are set to true. The instruction executes.
Instruction first run	N/A

Instruction first scan	Set request to initialize the Minimum value with the current input.
Postscan	EnableIn and EnableOut bits are cleared to false.

Structured Text

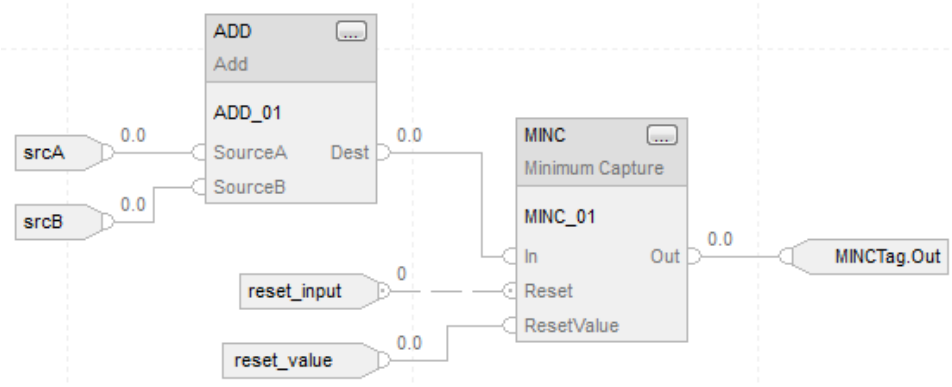
In Structured Text, EnableIn is always True during a normal scan. If the instruction is in the control path activated by the logic, it executes.

Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.
Normal Execution	See Tag.EnableIn is true in the Function Block table.
Postscan	See Postscan in the Function Block table.

Examples

If Reset is set, the instruction set Out=ResetValue. If Reset is cleared, the instruction set Out=In when In < LastMinimum. Otherwise, the instruction sets Out= LastMinimum.

Function Block



Structured Text

```
MINCTag.In := input_value;
MINCTag.Reset := reset_input;
MINCTag.ResetValue := reset_value;
MINC(MINCTag);
result := MINCTag.Out;
```

Moving Standard Deviation (MSTD)

This information applies to the CompactLogix 5370, ControlLogix 5570, CompactGuardLogix 5370, GuardLogix 5570, CompactGuardLogix 5380, CompactLogix 5380, ControlLogix 5580, GuardLogix 5580, and ControlLogix 5590 controllers.

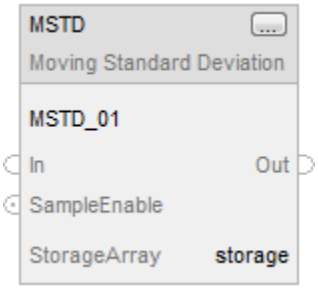
The Moving Standard Deviation (MSTD) instruction calculates a moving standard deviation and average for the In signal.

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram logic.

Function Block



Structured Text

MSTD(MSTD_tag, StorageArray);

Operands

Function Block

Operand	Type	Format	Description
block tag	MOVING_STD_DEV	structure	MSTD structure
StorageArray	REAL	array	Holds the In samples. This array must be at least as large as NumberOfSamples.

MOVING_STD_DEV Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
In	REAL	The analog signal input to the instruction. Valid = any float Default = 0.0
InFault	BOOL	Bad health indicator for the input. If In is read from an analog input, then

		InFault is normally controlled by fault status on the analog input. When set, InFault indicates the input signal has an error, the instruction sets the appropriate bit in Status, and the instruction holds Out and Average at their current values. When InFault transitions from set to cleared, the instruction initializes the averaging algorithm and continues executing. Default is cleared.
Initialize	BOOL	Initialize input to the instruction. When set, the instruction sets Out = 0.0 and Average = In, except when InFault is set, in which case, the instruction holds both Out and Average at their current values. When Initialize transitions from set to cleared, the instruction initializes the standard deviation algorithm and continues executing. Default is cleared.
SampleEnable	BOOL	Enable for taking a sample of In. When set, the instruction enters the value of In into the storage array and calculates a new Out and Average value. When SampleEnable is cleared and Initialize is cleared, the instruction holds Out and Average at their current values. Default is cleared.
NumberOfSamples	DINT	The number of samples to be used in the calculation. If this value is invalid, the instruction sets the appropriate bit in Status and the instruction holds Out and Average at their current values. When NumberOfSamples becomes valid again, the instruction initializes the standard deviation algorithm and continues executing. Valid = 1 to size of the storage array Default = 1

Output Parameter	Data Type	Description
EnableOut	BOOL	Indicates if instruction is enabled. Cleared to false if Out overflows

Out	REAL	The calculated output of the algorithm.
Average	REAL	The calculated average of the algorithm.
Status	DINT	Status of the function block.
InstructFault (Status.0)	BOOL	The instruction detected one of the following execution errors. This is not a minor or major controller error. Check the remaining status bits to determine what occurred.
InFaulted (Status.1)	BOOL	In health is bad. InFault is set.
NumberOfSampInv (Status.2)	BOOL	NumberOfSamples invalid or not compatible with array size.

Structured Text

Operand	Type	Format	Description
block tag	MOVING_STD_DEV	structure	MSTD structure
StorageArray	REAL	array	Holds the In samples. This array must be at least as large as NumberOfSamples.

See Structured Text Syntax on page for more information on the syntax of expressions within structured text.

Description

The MSTD instruction supports any input queue length. Each scan, if SampleEnable is set, the instruction enters the value of In into a storage array. When the storage array is full, each new value of In causes the oldest entry to be deleted.

The MSTD instructions uses these equations for the outputs:

Condition	Action
Average	$Average = \frac{\sum_{n=1}^{NumberOfSamples} In_n}{NumberOfSamples}$
Out	$Out = \sqrt{\frac{\sum_{n=1}^{NumberOfSamples} (In_n - Average)^2}{NumberOfSamples}}$

The instruction will not place an invalid In value (NAN or ± INF) into the storage array. When In is invalid, the instruction sets Out = In, sets Average = In, and logs an overflow minor fault, if this reporting is enabled. When In becomes valid, the instruction initializes the standard deviation algorithm and continues executing.

You can make runtime changes to the NumberOfSamples parameter. If you increase the number, the instruction incrementally processes new data from the current sample size to the new sample size. If you decrease the number, the instruction re-calculates the standard deviation from the beginning of the sample array to the new NumberOfSamples value.

Initializing the Standard Deviation Algorithm

Certain conditions, such as instruction first scan and instruction first run, require the instruction to initialize the standard deviation algorithm. When this occurs, the instruction considers the StorageArray empty and incrementally processes samples from 1 to the NumberOfSamples value. For example:

NumberOfSamples = 3

Scan 1: Average = $In_n/1$

Out = Square root $(((In_n\text{-Average})^2)/1)$

Scan 2: Average = $(In_n+In_{n-1})/2$

Out = Square root $(((In_n\text{-Average})^2+(In_{n-1}\text{-Average})^2)/2)$

Scan 3: Average = $(In_n+In_{n-1}+In_{n-2})/NumberOfSamples$

Out = Square root $(((In_n\text{-Average})^2+(In_{n-1}\text{-Average})^2+(In_{n-2}\text{-Average})^2)/NumberOfSamples)$

Affects Math Status Flags

No

Major/Minor Faults

None specific to this instruction. See Common Attributes on page 591 for operand-related faults.

Execution

Function Block

Condition/State	Action Taken
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is true	EnableIn and EnableOut bits are set to true. The instruction executes.
Instruction first run	Initialize the previous Output and Average.
Instruction first scan	Initialize Out to zero. Initialize Average to Input

	Initialize the instruction algorithm.
Postscan	EnableIn and EnableOut bits are cleared to false.

Structured Text

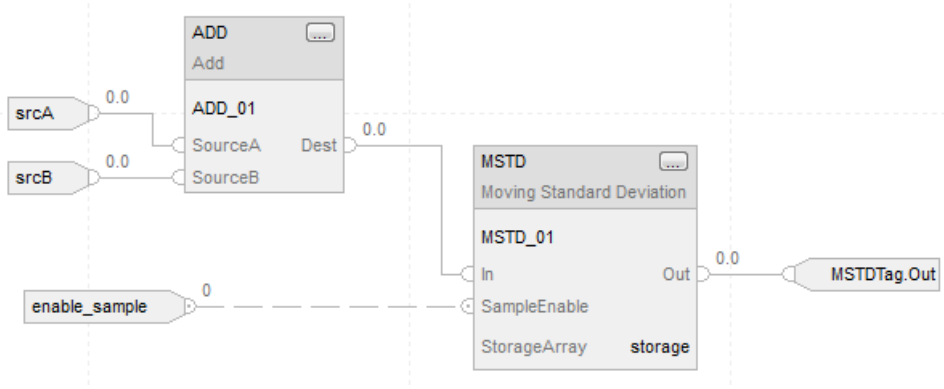
In Structured Text, EnableIn is always True during a normal scan. If the instruction is in the control path activated by the logic, it executes.

Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.
Normal Execution	See Tag.EnableIn is true in the Function Block table.
Postscan	See Postscan in the Function Block table.

Example

Each scan that SampleEnable is set, the instruction places the value of In into array storage, calculates the standard deviation of the values in array storage, and places the result in Out. Out becomes an input parameter for function_block_C.

Function Block



Structured Text

```
MSTD_01.In := input_value;
MSTD_01.SampleEnable := enable_sample;
MSTD(MSTD_01,storage);
deviation := MSTD_01.Out;
```

Logical and Move Instructions

These instructions are used to perform logical operations on and move output data.

Available Instructions

Ladder Diagram and Structured Text

DFF on page 455	JKFF on page 458	RESD on page 461	SETD on page 463

Function Block

Not available

Structured Text

DFF on page 455	JKFF on page 458	RESD on page 461	SETD on page 463

D Flip-Flop (DFF)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, Compact GuardLogix 5380, CompactLogix 5380, ControlLogix 5580, GuardLogix 5580, and ControlLogix 5590 controllers.

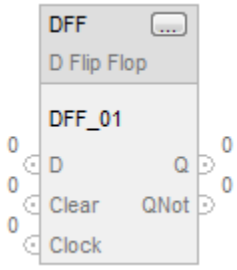
The DFF instruction sets the Q output to the state of the D input on a cleared to set transition of the Clock input. The QNot output is set to the opposite state of the Q output.

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram logic.

Function Block



Structured Text

DFF(DFF_tag);

Operands

Function Block

Operand	Type	Format	Description
DFF tag	FLIP_FLOP_D	structure	DFF structure

FLIP_FLOP_D Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
D	BOOL	The input to the instruction. Default is cleared.
Clear	BOOL	Clear input to the instruction. If set, the instruction clears Q and sets QNot.
Clock	BOOL	Clock input to the instruction. Default is cleared.

Output Parameter	Data Type	Description
EnableOut	BOOL	Indicates if instruction is enabled.
Q	BOOL	The output of the instruction.
QNot	BOOL	The complement of the Q output.

Structured Text

Operand	Type	Format	Description
DFF tag	FLIP_FLOP_D	structure	DFF structure

See *Structured Text Syntax* for more information on the syntax of expressions within structured text.

Description

When Clear is set, the instructions clears Q and sets QNot. Otherwise, if Clock is set and Clockn-1 is cleared, the instruction sets Q=D and sets QNot = NOT (D).

The instruction sets Clockn-1 = Clock state every scan.

Affects Math Status Flags

No

Major/Minor Faults

None specific to this instruction. See Common Attributes on page for operand-related faults.

Execution

Function Block

Condition/State	Action Taken
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is true	EnableIn and EnableOut bits are set to true. The instruction executes.
Instruction first run	Clockn-1 is set to 1. Qn-1 is cleared to 0.
Instruction first scan	Previous input Clock state is set True. Previous output Q state is set False.
Postscan	EnableIn and EnableOut bits are cleared to false.

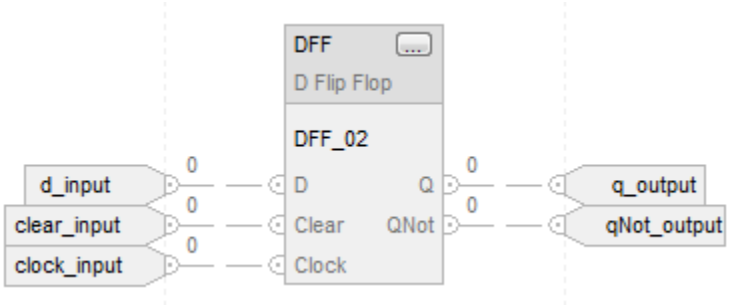
Structured Text

Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.
Normal Execution	See Tag.EnableIn is true in the Function Block table.
Postscan	See Postscan in the Function Block table.

Example

When Clock goes from cleared to set, the DFF instruction sets Q = D. When Clear is set, Q is cleared. The DFF instruction sets QNot to the opposite state of Q.

Function Block



Structured Text

```
DFF_03.D := d_input;
DFF_03.Clear := clear_input;
DFF_03.Clock := clock_input;
DFF(DFF_03);
q_output := DFF_03.Q;
qNot_output := DFF_03.QNot;
```

JK Flip-Flop (JKFF)

This information applies to the CompactLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, Compact GuardLogix 5380, CompactLogix 5380, ControlLogix 5580, GuardLogix 5580, and ControlLogix 5590 controllers.

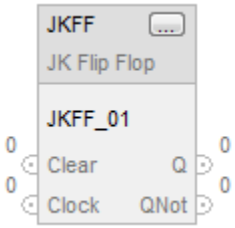
The JKFF instruction complements the Q and QNot outputs when the Clock input transitions from cleared to set.

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram logic.

Function Block



Structured Text

```
JKFF(JKFF_tag);
```

Operands

Function Block

Operand	Type	Format	Description
JKFF tag	FLIP_FLOP_JK	Structure	JKFF structure

FLIP_FLOP_JK Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
Clear	BOOL	Clear input to the instruction. If set, the instruction clears Q and sets QNot.
Clock	BOOL	Clock input to the instruction. Default is cleared.

Output Parameter	Data Type	Description
EnableOut	BOOL	Indicates if instruction is enabled.
Q	BOOL	The output of the instruction.
QNot	BOOL	The complement of the Q output.

Structured Text

Operand	Type	Format	Description
JKFF tag	FLIP_FLOP_JK	Structure	JKFF structure

See Structured Text Syntax for more information on the syntax of expressions within structured text.

Description

When Clear is set, the instructions clears Q and sets QNot. Otherwise, if Clock is set and Clockn-1 is cleared, the instruction toggles Q and QNot.

The instruction sets Clockn-1 = Clock state every scan.

Affects Math Status Flags

No

Major/Minor Faults

None specific to this instruction. See Common Attributes on page for operand-related faults.

Execution

Function Block

Condition/State	Action Taken
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is true	EnableIn and EnableOut bits are set to true. The instruction executes.
Instruction first run	Clockn-1 is set to 1. Qn-1 is cleared to 0.
Instruction first scan	Previous input Clock states is set to True. Previous output Q state is False.
Postscan	EnableIn and EnableOut bits are cleared to false.

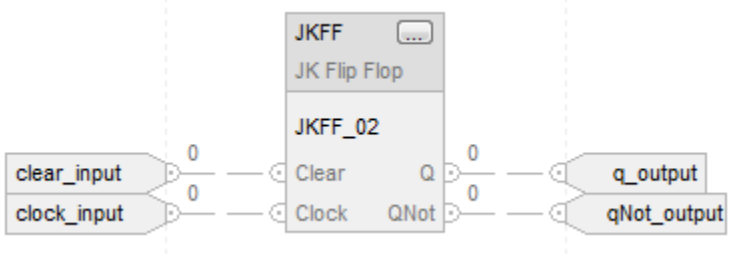
Structured Text

Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.
Normal Execution	See Tag.EnableIn is true in the Function Block table.
Postscan	See Postscan in the Function Block table.

Examples

When Clock goes from cleared to set, the JKFF instruction toggles Q. If Clear is set, Q is always cleared. The JKFF instruction sets QNot to the opposite state of Q.

Function Block



Structured Text

```
JKFF_01.Clear := clear_input;
JKFF_01.Clock := clock_input;
JKFF(JKFF_01);
q_output := JKFF_01.Q;
qNot_output := JKFF_01.QNot;
```

Reset Dominant (RESD)

This information applies to the Compact GuardLogix 5370, ControlLogix 5570, Compact GuardLogix 5370, GuardLogix 5570, Compact GuardLogix 5380, ControlLogix 5580, and ControlLogix 5590 controllers.

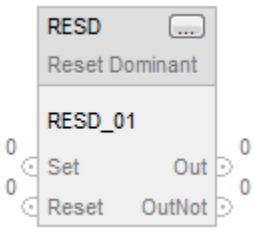
The RESD instruction uses Set and Reset inputs to control latched outputs. The Reset input has precedence over the Set input.

Available Languages

Ladder Diagram

This instruction is not available for ladder diagram.

Function Block



Structured Text

RESD(RESD_tag);

Operands

Function Block

Operand	Type	Format	Description
RESD tag	DOMINANT_RESET	structure	RESD structure

DOMINANT_RESET Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated.

		Default is set.
Set	BOOL	Set input to the instruction. Default is cleared.
Reset	BOOL	Reset input to the instruction. Default is cleared.

Output Parameter	Data Type	Description
EnableOut	BOOL	Indicates if instruction is enabled.
Out	BOOL	The output of the instruction.
OutNot	BOOL	The inverted output of the instruction.

Structured text

See *Structured Text Syntax* for more information on the syntax of expressions within structured text.

Description

The Reset Dominant instruction uses the Set and Reset input parameters to control latched output parameters Out and OutNot. The Reset input has precedence over the Set input.

Out will be latched true whenever the Set input parameter is set true. Setting the Reset parameter to true will override the current state of Out, setting Out to false.

When Reset returns to false, Out will be latched to the current state of the Set input parameter. OutNot will be set to the opposite state of Out.

Affects Math Status Flags

No

Major/Minor Faults

None specific to this instruction. See Common Attributes on page for fault related attributes.

Execution

Function Block

Condition/State	Action Taken
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is true	EnableIn and EnableOut bits are set to true. The instruction executes.

Instruction first run	Out bit is set to false. OutNot is set to true.
Instruction first scan	N/A
Postscan	EnableIn and EnableOut bits are cleared to false.

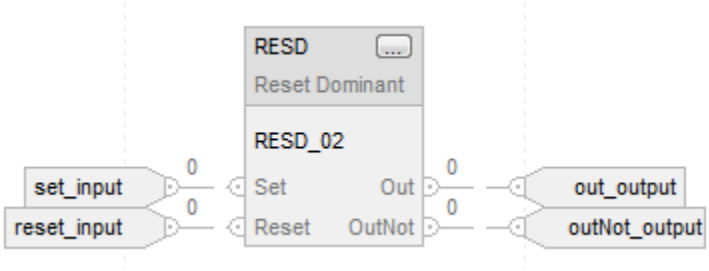
Structured Text

Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.
Normal Execution	See Tag.EnableIn is true in the Function Block table.
Postscan	See Postscan in the Function Block table.

Example

When Set is true and Reset is false, Out is set true. When Reset is true, Out is cleared. The Reset input has precedence over the Set input. The RESD instruction sets OutNot to the opposite state of Out.

Function Block



Structured Text

```
RESD_01.Set := set_input;
RESD_01.Reset := reset_input;
RESD(RESD_01);
out_output := RESD_01.Out;
outNot_output := RESD_01.OutNot;
```

Set Dominant (SETD)

This instruction applies to the CompactLogix 5370, ControlLogix 5570, ControlLogix 5580, and ControlLogix 5590 controllers.

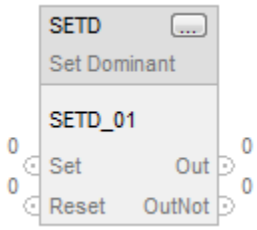
The SETD instruction uses Set and Reset inputs to control latched outputs. The Set input has precedence over the Reset input.

Available Languages

Ladder Diagram

This instruction is not available in ladder diagram logic.

Function Block



Structured Text

SETD(SETD_tag);

Operands

Function Block

Operand	Type	Format	Description
SETD tag	DOMINANT_SET	structure	SETD structure

Structured text

Operand	Type	Format	Description
SETD tag	DOMINANT_SET	structure	SETD structure

See *Structured Text Syntax* for more information on the syntax of expressions within structured text.

DOMINANT_SET Structure

Input Parameter	Data Type	Description
EnableIn	BOOL	Enable input. If cleared, the instruction does not execute and outputs are not updated. Default is set.
Set	BOOL	Set input to the instruction. Default is cleared.
Reset	BOOL	Reset input to the instruction. Default is cleared.

Output Parameter	Data Type	Description
EnableOut	BOOL	Indicates if instruction is enabled.
Out	BOOL	The output of the instruction.
OutNot	BOOL	The inverted output of the instruction.

Description

The Set Dominant instruction uses the Set and Reset input parameters to control latched output parameters Out and OutNot. The Set input has precedence over the Reset input.

Out will be latched true whenever the Set input parameter is set true. Setting the Reset parameter to true will set Out to false only if the Set input is false. OutNot will be set to the opposite state of Out.

Affects Math Status Flags

No

Major/Minor Faults

None specific to this instruction. See Common Attributes on page for operand-related faults.

Execution

Condition/State	Action Taken
Prescan	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is false	EnableIn and EnableOut bits are cleared to false.
Tag.EnableIn is true	EnableIn and EnableOut bits are set to true. The instruction executes.
Instruction first run	Out bit is set to true. OutNot is cleared to false.
Instruction first scan	N/A
Postscan	EnableIn and EnableOut bits are cleared to false.

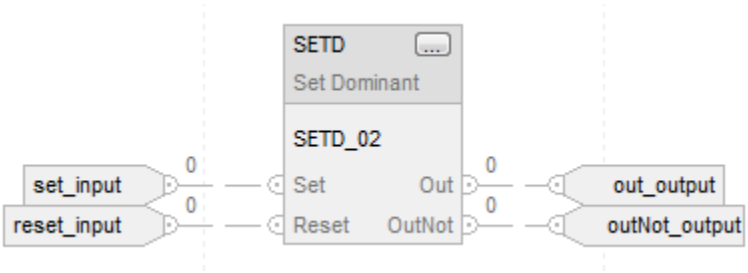
Structured Text

Condition/State	Action Taken
Prescan	See Prescan in the Function Block table.
Normal Execution	See Tag.EnableIn is true in the Function Block table.
Postscan	See Postscan in the Function Block table.

Example

When Set is true, Out is set true. When Set is false and Reset is true, Out is cleared. The Set input has precedence over the Reset input. The SETD instruction sets OutNot to the opposite state of Out.

Function Block



Structured Text

```
SETD_01.Set := set_input;
SETD_01.Reset := reset_input;
SETD(SETD_01);
out_output := SETD_01.Out;
outNot_output := SETD_01.OutNot;
```

Equipment Phase Instructions

The Equipment Phase instructions include:

Available Instructions

Ladder Diagram and Structured Text

PSC on page 514	PCMD on page 477	POVR on page 506	PFL on page 498	PCLF on page 475	PXRQ on page 484	PRNP on page 503	PPD on page 510	PATT on page 467	PDET on page 473

Function Block

These instructions are not available in function block.

If want to:	Use this instruction:
Signal an equipment phase that the state routine is complete to indicate that it should go to the next state	PSC
Change the state or substate of an equipment phase	PCMD
Give a Hold, Stop, or Abort command to an equipment phase, regardless of ownership	POVR
Signal a failure for an equipment phase	PFL
Clear the failure code of an equipment phase	PCLF
Initiate communication with FactoryTalk Batch software.	PXRQ
Clear the NewInputParameters bit of an equipment phase	PRNP
Set up breakpoints within the logic of an equipment phase	PPD
Take ownership of an equipment phase to either: • Prevent another program or FactoryTalk Batch software from commanding an equipment phase • Make sure another program or FactoryTalk Batch software does not already own an equipment phase	PATT
Relinquish ownership of an equipment phase	PDET

Attach to Phase (PATT)

This instruction applies to the Compact GuardLogix 5370, Compact GuardLogix 5380, CompactLogix 5370, CompactLogix 5380, GuardLogix 5570, GuardLogix 5580, ControlLogix 5570, ControlLogix 5580, and ControlLogix 5590 controllers.

When using this instruction with ControlLogix redundancy system, outputs controlled by this instruction may not be bumpless during redundancy switchover, if the instruction and phase program are not scheduled in the highest priority task.

Use the PATT instruction to take ownership of an equipment phase to either:

- Prevent another program or FactoryTalk Batch software from commanding an equipment phase.
- Make sure another program or FactoryTalk Batch software does not already own an equipment phase.

The PATT instruction lets a program take ownership of an equipment phase.

- Ownership is optional. As long as an equipment phase has no owners, any sequencer (program in the controller, FactoryTalk Batch software) can command an equipment phase.
- FactoryTalk Batch software always takes ownership of an equipment phase.
- Once a sequencer owns an equipment phase, no other sequencer can command the equipment phase.

This is a transitional instruction. Follow these steps when using it:

- In ladder logic, insert an instruction to toggle the rung-condition-in from false to true each time the instruction should execute.
- In a Structured Text routine, insert a condition for the instruction to cause it to execute only on a transition.

Available Languages

Ladder Diagram

Function Block

This instruction is not available in function block.

Structured Text

PATT(Phase_Name,Result);

Operands

Ladder Diagram

Operand	Type	Format	Description
Phase Name	PHASE	Name of the equipment phase	Equipment phase to own.
Result	DINT	immediate tag	To let the instruction return a code for its success or failure, enter a DINT tag in which to store the result code. Otherwise, enter 0.

Structured Text

The operands are the same as those for the Ladder Diagram PATT instruction.

Guidelines for using the PATT Instruction

Guideline	Details	
Consider ownership if have multiple sequencers that use a common equipment phase.	Ownership makes sure that a program can command all the equipment phases it needs and locks out any other sequencers.	
	If using:	Then:
	FactoryTalk Batch software to also run sequences within this controller	Before executing the sequence (process), take ownership of all the equipment phases that the sequence uses.
	Multiple programs to command the same equipment phase	
	None of the above	No need to own the equipment phases.
Remember that Logix Designer overrides the controller.	Regardless of whether a program or FactoryTalk Batch software owns an equipment phase, the option exists to use Logix Designer to override ownership and command the equipment phase to a different state.	
	This:	Overrides this:
	Logix Designer	Controller (internal sequencer), FactoryTalk Batch software (external sequencer)
	Controller (internal sequencer)	None

Guideline	Details	
	FactoryTalk Batch software (external sequencer)	None
Use the Result operand to validate ownership.	Use the Result operand to get a code that shows the success or failure of the PATT instruction. To interpret the result code, see the *PATT Result Codes* section below.	
Avoid or plan for a result code = 24582.	On each execution, the PATT instruction tries to take ownership of the equipment phase. Once a program owns an equipment phase, another execution of the PATT instruction produces a result code = 24582. When using a PATT instruction, either: • Limit its execution to a single scan to avoid the 24582 result code. • Include a result code = 24582 in conditions for ownership.	
When the sequence is done, relinquish ownership.	To relinquish ownership, use a Detach from Equipment Phase (PDET) instruction.	

PATT Result Codes

If assigning a tag to store the result of a PATT instruction, the instruction returns one of these codes when it executes:

Code (Dec)	Description
0	The command was successful.
24579	Logix Designer or HMI already owns equipment phase. The caller is attached to the equipment phase, but it is not the current commanding owner. • This program now also owns the equipment phase. • Since the Logix Designer or HMI is higher priority than a program, the program cannot command the equipment phase. • High priority HMI ownership is specific only to CompactLogix 5370 and ControlLogix 5570 controllers.
24582	The program already owns the equipment phase.
24593	One of these already owns the equipment phase: • External sequencer (FactoryTalk Batch software) • Another program in the controller
24594	The equipment phase is unscheduled, inhibited, or in a task that is inhibited.

Affects Math Status Flags

No

Major/Minor Faults

None. See *Index Through Arrays* for operand-related faults.

Execution

For Structured Text, EnableIn is always true during normal scan. Therefore, if the instruction is in the control path activated by the logic, it will execute. All Conditions below the thick solid line can only occur during Normal Scan mode.

Condition/State	Action Taken
Prescan	No action taken.
Postscan	No action taken.
EnableIn is false	No action taken.
EnableIn is true	The instruction executes as described above.

Example

Ladder Diagram

If *Step.1* = 1 (first step in the sequence) then

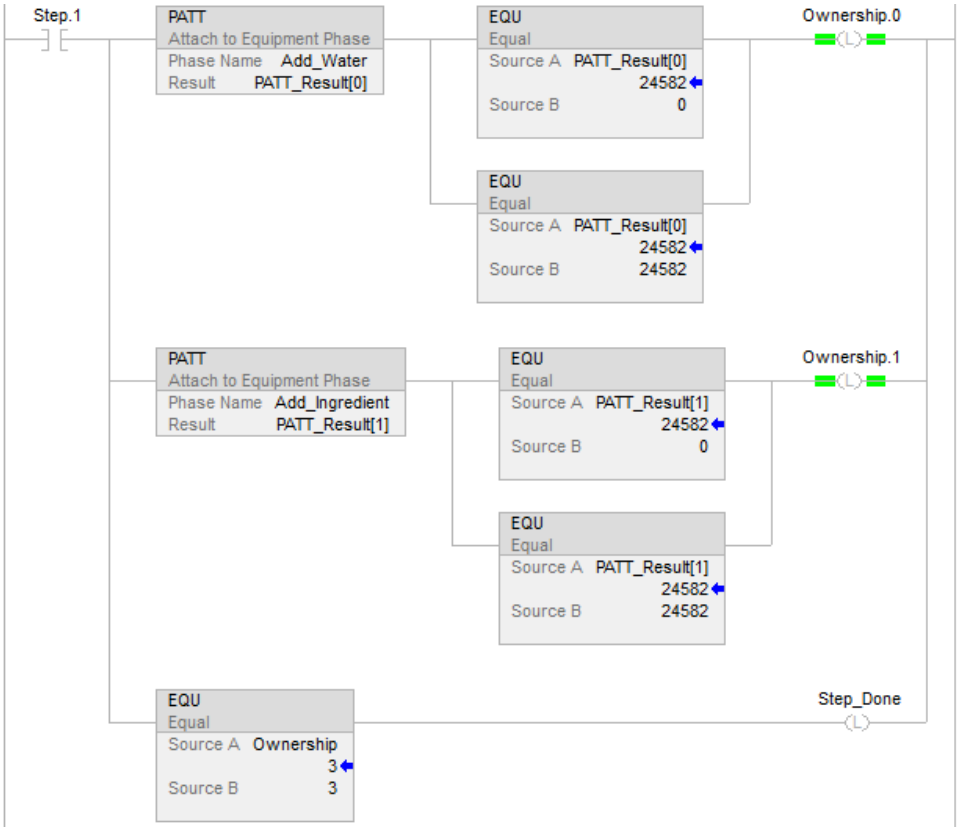
Each PATT instruction tries to take ownership of an equipment phase.

If the Result of a PATT instruction = 0 or 24582 (the program owns the equipment phase), then

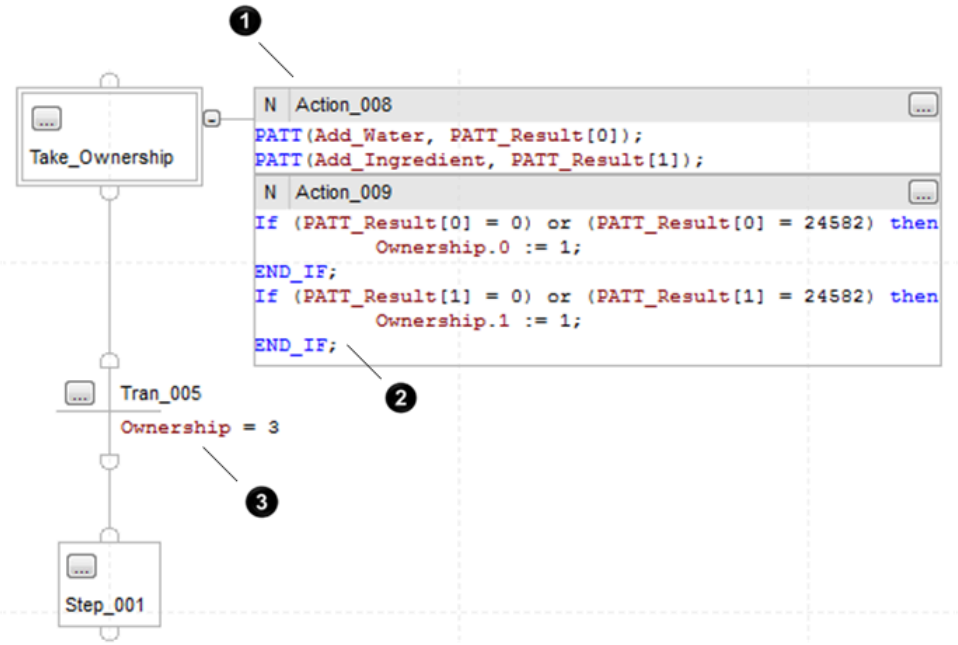
A bit within the *Ownership* tag = 1. (In the *Ownership* tag, each equipment phase is assigned a bit.)

If *Ownership* = 3 (The program owns both equipment phases as shown by bits 0 and 1), then

Done = 1. (This signals the sequence to go to the next step.)



Structured Text



Number	Description
1	At the first step in the sequence, the Take_Ownership action tries to take ownership of two equipment phases that the sequence uses.
2	Action_009 checks that the program owns the equipment phases. If the Result of each PATT instruction = 0 or 24282 (the program owns the equipment phase), then a bit within the Ownership tag = 1. (In the Ownership tag, each equipment phase is assigned a bit.)
3	If the Ownership = 3 (The program owns both equipment phases as shown by bits 0 and 1.), then the SFC goes to the next step.

Detach from Phase (PDET)

This instruction applies to the Compact GuardLogix 5370, Compact GuardLogix 5380, CompactLogix 5370, CompactLogix 5380, GuardLogix 5570, GuardLogix 5580, ControlLogix 5570, ControlLogix 5580, and ControlLogix 5590 controllers.

When using this instruction with ControlLogix redundancy system, outputs controlled by this instruction may not be bumpless during redundancy switchover, if the instruction and phase program are not scheduled in the highest priority task.

Use the PDET instruction to relinquish ownership of an equipment phase.
After a program executes a PDET instruction, the program *no longer* owns the equipment phase. This frees the equipment phase for ownership by another program or by FactoryTalk Batch software. Use the PDET instruction only if the program previously took ownership of an equipment phase via an Attach to Equipment Phase (PATT) instruction.

This is a transitional instruction. Follow these steps when using it:

- In ladder logic, insert an instruction to toggle the rung-condition-in from false to true each time the instruction should execute.
- In a Structured Text routine, insert a condition for the instruction to cause it to execute only on a transition.

Available Languages

Ladder Diagram

Function Block

This instruction is not available in function block.

Structured Text

```
PDET(Phase_Name);
```

Operands

Ladder Diagram

Operand	Type	Format	Description
Phase Name	PHASE	Name of the equipment phase	Equipment phase no longer to own.

Structured Text

The operands are the same as those for the Ladder Diagram PDET instruction.

Affects Math Status Flags

No

Major/Minor Faults

None. See *Index Through Arrays* for operand-related faults.

Execution

For Structured Text, EnableIn is always true during normal scan. Therefore, if the instruction is in the control path activated by the logic, it will execute.

Condition/State	Action Taken
Prescan	No action taken.
Postscan	No action taken.
EnableIn is false	No action taken.
EnableIn is true	The instruction executes.

Example

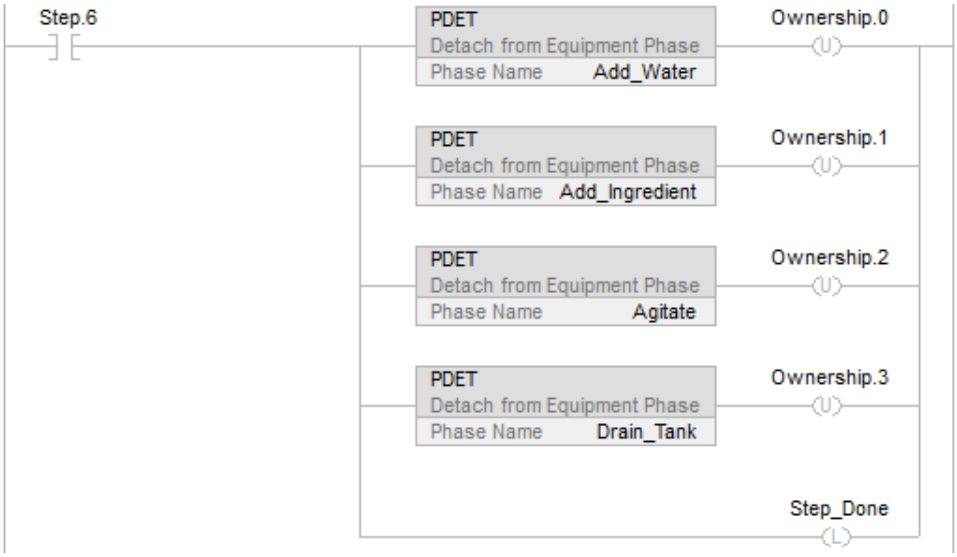
Ladder Diagram

If *Step.6* = 1 (step 6 in the sequence) then

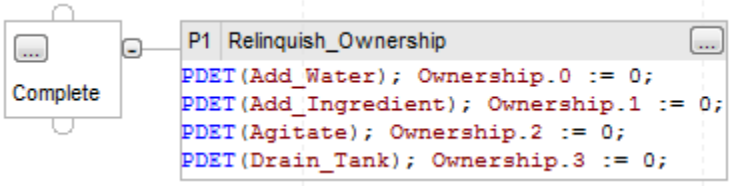
Each PDET instruction relinquishes ownership of the phases that the sequence owned.

Each *Ownership* bit = 0. (In the *Ownership* tag, each equipment phase is assigned a bit.)

Done = 1. (This signals the sequence to go to the next step.)



Structured Text



When the sequence executes, the Relinquish_Ownership action:

- Relinquishes ownership of the equipment phase.
- Clears the ownership flags (bits that the SFC set when it took ownership of the equipment phases).

Using an action Qualifier of type P1 limits the action execution to the first scan of that step.

Phase Clear Failure (PCLF)

This instruction applies to the Compact GuardLogix 5370, Compact GuardLogix 5380, CompactLogix 5370, CompactLogix 5380, GuardLogix 5570, GuardLogix 5580, ControlLogix 5570, ControlLogix 5580, and ControlLogix 5590 controllers.
Use the PCLF instruction to clear the failure code of an equipment phase.

The PCLF instruction clears the failure code for an equipment phase.

- Use only a PCLF instruction to clear the failure code of an equipment phase.
- A CLR instruction, MOV instruction, or assignment (:=) *does not* change the failure code of an equipment phase.

Make sure the equipment phase *does not* have other owners when using the PCLF instruction. The PCLF instruction *will not* clear the failure code if Logix Designer, HMI, FactoryTalk Batch software, or another program owns the equipment phase.

- High priority HMI ownership is specific only to the CompactLogix 5370 and ControlLogix 5570 controllers.

When using this instruction with ControlLogix redundancy system, outputs controlled by this instruction may not be bumpless during redundancy switchover, if the instruction and phase program are not scheduled in the highest priority task.

This is a transitional instruction. Follow these steps when using it:

- In ladder logic, insert an instruction to toggle the rung-condition-in from false to true each time the instruction should execute.
- In a Structured Text routine, insert a condition for the instruction to cause it to execute only on a transition.

Available Languages

Ladder Diagram

Function Block

This instruction is not available in function block.

Structured Text

PCLF(Phase_Name);

Operands

Ladder Diagram

Operand	Type	Format	Description
Phase Name	PHASE	Name of the equipment phase	Equipment phase whose failure code to clear.

Structured Text

The operands are the same as those for the Ladder Diagram PCLF instruction.

Affects Math Status Flags

No

Major/Minor Faults

None. See *Index Through Arrays* below for operand-related faults.

Execution

For Structured Text, EnableIn is always true during normal scan. Therefore, if the instruction is in the control path activated by the logic, it executes.

Condition/State	Action Taken
Prescan	No action taken.
Postscan	No action taken.
EnableIn is false	No action taken.
EnableIn is true	The instruction executes as described above.

Example

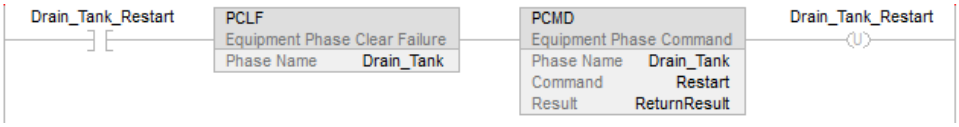
Ladder Diagram

If *Drain_Tank_Restart* = 1 (restart the *Drain_Tank* equipment phase) then

Clear the failure code of the *Drain_Tank* equipment phase

Change the state of the *Drain_Tank* equipment phase to restarting via the restart command.

Drain_Tank_Restart = 0;



Structured Text

(*If *Drain_Tank_Restart* = on, then

Clear the failure code for the *Drain_Tank* equipment phase.

Restart the *Drain_Tank* equipment phase.

Turn off *Drain_Tank_Restart*.*)

If *Drain_Tank_Restart* Then

PCLF(*Drain_Tank*);

PCMD(*Drain_Tank*,Restart,0);

Drain_Tank_Restart := 0;

End_If;

Phase Command (PCMD)

This instruction applies to the Compact GuardLogix 5370, Compact GuardLogix 5380, CompactLogix 5370, CompactLogix 5380, GuardLogix 5570, GuardLogix 5580, ControlLogix 5570, ControlLogix 5580, and ControlLogix 5590 controllers.

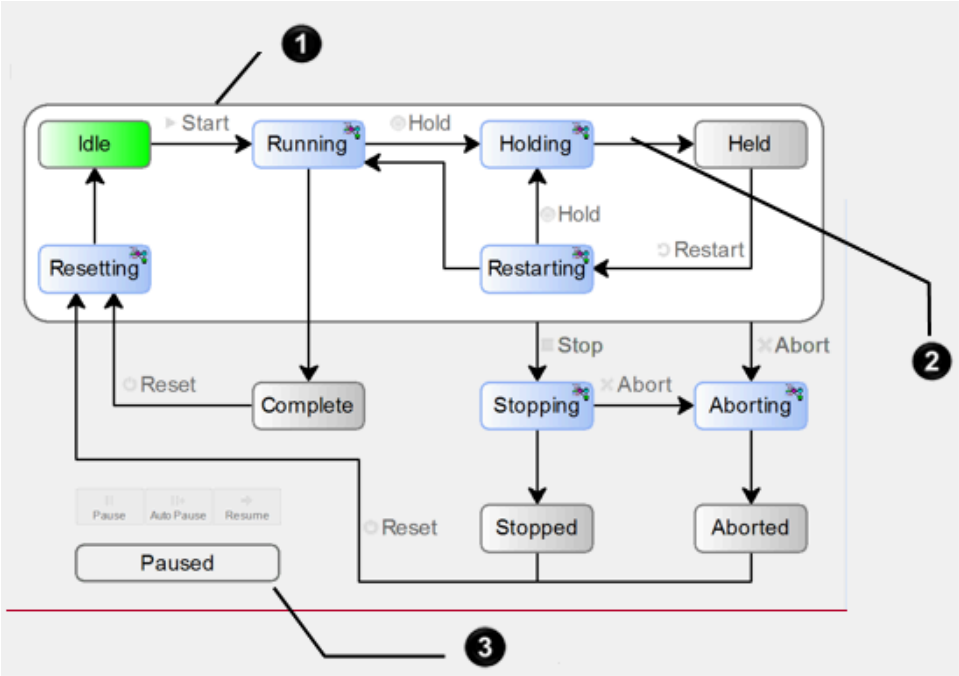
When using this instruction with ControlLogix redundancy system, outputs controlled by this instruction may not be bumpless during redundancy switchover, if the instruction and phase program are not scheduled in the highest priority task.

The PCMD instruction transitions an equipment phase to the next state or sub-state.
Use the PCMD instruction to change the state or sub-state of an equipment phase.

In the running state routine, use the PSC instructions to transition the equipment phase to the complete state. For more information about Paused functionality, please see the PPD phase instruction.

- This is a transitional instruction. Follow these steps when using it:
  - In ladder logic, insert an instruction to toggle the rung-condition-in from false to true each time the instruction should execute.
  - In a Structured Text routine, insert a condition for the instruction to cause it to execute only on a transition.

The PPD instruction is necessary for using pause functionality.



Number	Description
1	Command Some states need a command to go to the next state. If the equipment phase is in the idle state, a start command transitions the equipment phase to the running state. Once in the running state, the equipment phase executes its running state routine.
2	Other states use a *Phase State Complete (PSC)* instruction to go to the next state. If the equipment phase is in the holding state, a PSC instruction transitions the equipment phase to the held state. Once in the held state, the equipment phase needs a restart command to go to the restarting state.
3	Sub-state Use Auto Pause, Pause, and Resume to test and debug a state routine. The sub-states require the *Equipment*

Number	Description
	Phase Paused (PPD) instruction to create breakpoints in the logic. Use the auto pause, pause, and resume commands to step through the breakpoints.

Available Languages

Ladder Diagram

Function Block

This instruction is not available in function block.

Structured Text

PCMD (PhaseName,Command,Result);

Operands

Ladder Diagram

Operand	Type	Format	Description
Phase Name	PHASE	Name of the equipment phase	Equipment phase to change to a different state.
Command	command	Command enumeration value	Command to send to the equipment phase to change its state. For available commands, refer to the illustration above.
Result	DINT	immediate tag	To let the instruction return a code for its success or failure, enter a DINT tag in which to store the result code. Otherwise, enter 0.

Structured Text

The operands are the same as those for the Ladder Diagram PCMD instruction.

Guidelines for using the PCMD Instruction

<table>
<tr><th>Guideline</th><th colspan="2">Details</th></tr>
<tr><td>Limit execution of a PCMD instruction to a single scan.</td><td colspan="2">Limit the execution of the PCMD instruction to a single scan. Each command applies to only a specific state or states. Once the equipment phase changes state, the command is no longer valid. To limit execution, use methods such as:
• Execute the PCMD instruction within a P1 Pulse (Rising Edge) or P0 Pulse (Falling Edge) action.
• Place a one shot instruction before the PCMD instruction.
• Execute the PCMD instruction and then advance to the next step.</td></tr>
<tr><td rowspan="4">Determine if need ownership of the equipment phase.</td><td colspan="2">As an option, a program can own an equipment phase. This prevents another program or FactoryTalk Batch software from also commanding the equipment phase.</td></tr>
<tr><td>If using:</td><td>Then:</td></tr>
<tr><td>FactoryTalk Batch software to also run procedures (recipes) within this controller</td><td>Also run procedures (recipes) within this controller. Before using a PCMD instruction, use an Attach to Equipment Phase (PATT) instruction to take ownership of the equipment phase.</td></tr>
<tr><td>Multiple programs to command the same equipment phase</td><td>Use an Attach to Equipment Phase (PATT) instruction to take ownership of the equipment phase.</td></tr>
<tr><td></td><td>None of the above</td><td>There is no need to own the equipment phase.</td></tr>
<tr><td>Using a POVR instruction instead of a PCMD instruction.</td><td colspan="2">1. Giving the hold, stop, or abort command?
- No - Use the PCMD instruction.
- Yes - Go to step 2.
2. Must the command work even if the equipment phase is under manual control via Logix Designer?
- Yes - Use the POVR instruction instead.
- No - Go to step 3.
3. Must the command work even if FactoryTalk Batch software or another program owns the equipment phase?
- Yes - Use the POVR instruction instead.
- No - Use the PCMD instruction.
For example, suppose the equipment checks for jammed material. And if there is a jam, the equipment should always abort. In that case, use the POVR instruction. This way, the equipment aborts even if under manual control via Logix Designer.</td></tr>
</table>

Guideline	Details	
Determine if need a return code.	Use the Result operand to get a code that shows the success or failure of the PCMD instruction.	
	If:	Then in the Result operand, enter a:
	Anticipate ownership conflicts or other possible errors	DINT tag, in which to store a code for the result of the execution of the instruction.
	Do *not* anticipate ownership conflicts or other possible errors	0
	Want to interpret the result code, refer to the *Result Codes* table below.	

PCMD Result Codes

If assigning a tag to store the result of a PCMD instruction, the instruction returns one of these codes when it executes:

Code (Dec)	Description
0	Successful command.
24577	Valid command.
24578	Invalid command for the current state of the equipment phase. For example, if the equipment phase is in the running state, then a start command is not valid.
24579	Cannot command the equipment phase. One of these already owns the equipment phase. • Logix Designer • HMI • external sequencer (for example, FactoryTalk Batch software) • another program in the controller
24594	Unscheduled or inhibited equipment phase or in an inhibited task.

High priority HMI ownership is specific only to CompactLogix 5370 and ControlLogix 5570 controllers.

Affects Math Status Flags

No

Major/Minor Faults

None. See *Index Through Arrays* for operand-related faults.

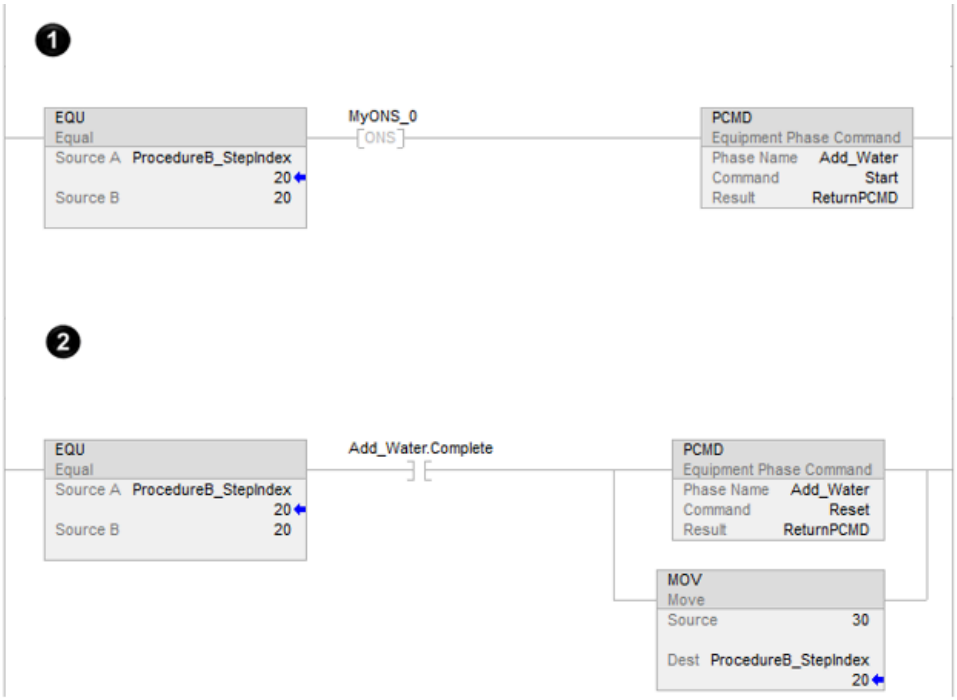
Execution

For Structured Text, EnableIn is always true during normal scan. Therefore, if the instruction is in the control path activated by the logic, it will execute. All conditions below the think solid line can only occur during Normal Scan mode.

Condition/State	Action Taken
Prescan	No action taken.
Postscan	No action taken.
EnableIn is false	No action taken.
EnableIn is true	The instruction executes as described above.

Example 1

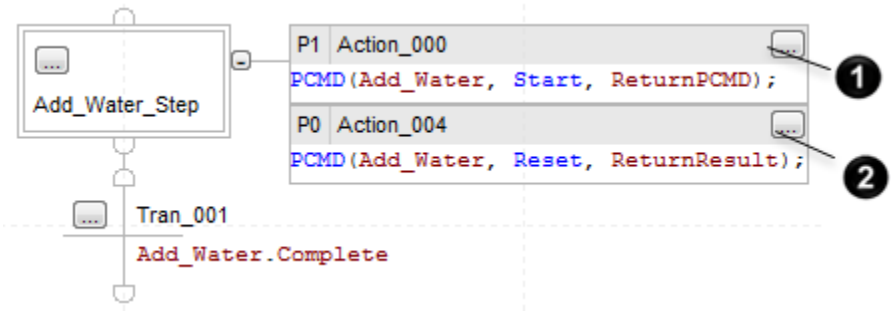
Ladder Diagram



Number	Description
1	If ProcedureB_Stepindex = 20 (the routine is at step 20) And this is the transition to step 20 (the ONS instruction signals that the EQU instruction went from false to true.) Then

Number	Description
	Change the state of the Add_Water equipment phase to running via the start command.
2	If ProcedureB_Stepindex = 20 (the routine is at step 20) And the Add_Water equipment phase is complete (Add_Water.Complete = 1) Then Change the state of the Add_Water equipment phase to resetting via the reset command. Advance to step 30.

Structured Text



Number	Description
1	When the SFC enters *Add_Water_Step*, change *ADD_Water* equipment phase to running via the start command, The P1 qualifier limits this to the first scan of the step.
2	Before the SFC leaves *Add_Water_Step* (*Add_Water_Complete* = 1), change *Add_Water* equipment phase to resetting via the reset command. The P0 qualifier limits this to the last scan of the step.

Example 2

Ladder Diagram

If ProcedureB_Stepindex = 30 (the routine is at step 30)

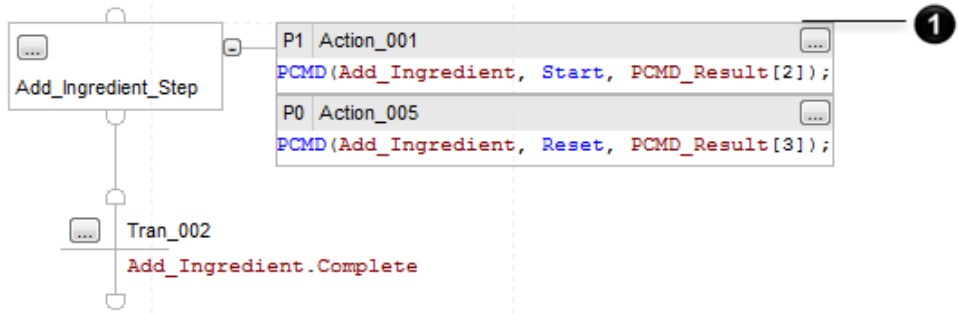
And this is the transition to step 30 (the ONS instruction signals that the EQU instruction went from false to true.)

Then

Change the state of the Add_Water equipment phase to running via the start command.

Verify that the command was successful and store the result code in PCMD_Result[1] [DINT Tag].

Structured Text



When the SFC enters *Add_Ingredient_Step*

- Change *Add_Ingredient* equipment phase to running via the start commend.
- Verify that the command was successful and store the result code in *PCDM_Result[2]* (DINT tag).

The P1 qualifier limits this to the first scan of the step.

Phase External Request (PXRQ)

This instruction applies to the Compact GuardLogix 5370, Compact GuardLogix 5380, CompactLogix 5370, CompactLogix 5380, GuardLogix 5570, GuardLogix 5580, ControlLogix 5570, ControlLogix 5580, and ControlLogix 5590 controllers.

When using this instruction with ControlLogix redundancy system, outputs controlled by this instruction may not be bumpless during redundancy switchover, if the instruction and phase program are not scheduled in the highest priority task.

Use the PXRQ instruction to initiate communication with FactoryTalk® Batch software.

IMPORTANT: When the PXRQ instruction is used in an Equipment Sequence, only the Download All (1000) request and the Upload All (2000) request are supported. All other PXRQ instruction requests are ignored.

The PXRQ instruction sends a request to FactoryTalk Batch software.

This is a transitional instruction. Follow these steps when using it:

- In ladder logic, insert an instruction to toggle the rung-condition-in from false to true each time the instruction should execute.
- In a Structured Text routine, insert a condition for the instruction to cause it to execute only on a transition.

Available Languages

Ladder Diagram

Function Block

This instruction is not available in function block.

Structured Text

PXRQ (Phase_Instruction, External_Request, Data_Value);

Operands

Ladder Diagram

Operand	Type	Format	Description
Phase Instruction	PHASE_INSTRUCTION	tag	Tag that controls the operation.
External Request	request	enumeration value	Type of request.
Data Value	DINT	array tag	Parameters of the request.

Structured Text

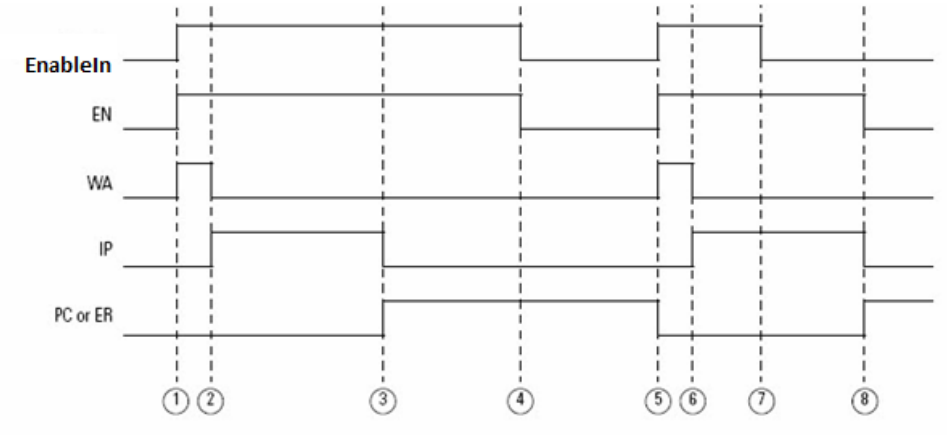
The operands are the same as those for the Ladder Diagram PXRQ instruction.

PHASE_INSTRUCTION Data Type

If want to:	Then check or set this member:	Data type	Notes
Determine if a false-to-true transition caused the instruction to execute	EN	BOOL	Refer to the timing diagram below.
Determine if the request failed	ER	BOOL	Refer to the timing diagram below. To diagnose the error, refer to the ERR and EXERR values.
Determine if the FactoryTalk Batch software completed	PC	BOOL	Refer to the timing diagram below.

If want to:	Then check or set this member:	Data type	Notes	
its processing of the request				
Determine if the FactoryTalk Batch software is processing the request	IP	BOOL	Refer to the timing diagram below.	
Determine if the instruction sent the request but FactoryTalk Batch software has not yet acknowledged it	WA	BOOL	Refer to the timing diagram below. WA also = 0 if: • The connection times out • A network error occurs • ABORT = 1	
Cancel the request	ABORT	BOOL	To abort (cancel) the request, set the ABORT bit = 1. When the controller aborts the instruction: • ER = 1 • ERR shows the result of the abort	
• Diagnose the cause of an error	ERR	INT	If ER = 1, the error code gives diagnostic information. To interpret the error code, see *PXRQ Error Codes*.	
• Write logic to respond to specific errors	EXERR	INT	If ER = 1, the extended error code gives additional diagnostic information for some errors. To interpret the extended error code, see *PXRQ Error Codes*.	
Use one member for the various status bits of the tag	STATUS	DINT	For this member:	Use this bit:
			EN	31
			ER	28
			PC	27
			IP	26
			WA	25
			ABORT	24

Timing Diagram



Guidelines for using the PXRQ Instruction

Guideline	Details
Make sure to use an array for the Data Values operand.	The Data Values operand requires a DINT array, even if the array contains only 1 element (that is, the data type is DINT[1]).
In Ladder Diagram, condition the instruction to execute on a transition.	This is a transitional instruction. Each time the instruction executes, toggle the EnableIn from false to true
In Structured Text, use a construct to condition the execution of the instruction.	When programming a PXRQ instruction in structured text, consider: • In structured text, instructions execute *each time* they are scanned. • The PXRQ instruction updates its status bits *only* when it is scanned. • To keep the instruction from repeatedly executing but ensure that the status bits update, enclose the instruction in a construct that: - Initiates the execution of the instruction *only* on a transition (change in conditions). - Remains true until either PC = 1 or ER = 1

Configure the PXRQ Instruction

If you want to:	Then configure the PXRQ instruction as follows:		
	External request	**Data Value Array Element**	**Value**
Download all input parameters	Download Input Parameters	DINT[0]	0

If you want to:	Then configure the PXRQ instruction as follows:		
Download a single input parameter	Download Input Parameters	DINT[0]	parameter ID
Download a range of input parameters	Download Input Parameters	DINT[0]	parameter ID of the first parameter
		DINT[1]	Number of parameters to download
Download the input parameters configured for automatic download on start or transfer of control	Download Input Parameters Subset	DINT[0]	start = 1 transfer of control = 2
Download all output parameters	Download Output Parameter Limits	DINT[0]	0
Download a single output parameter	Download Output Parameter Limits	DINT[0]	parameter ID
Upload all reports	Upload Output Parameters	DINT[0]	0
Upload a single report	Upload Output Parameters	DINT[0]	report ID
Upload a range of reports	Upload Output Parameters	DINT[0]	report ID of the first report
		DINT[1]	Number of reports to download
Upload the output parameters configured for automatic upload on terminal state or transfer of control	Upload Output Parameters Subset	DINT[0]	terminal = 1 transfer of control = 2
Send a message to an operator	Send Message to Operator	DINT[0]	message ID
Clear a message from an operator	Clear Message to Operator	DINT[0]	0
Acquire a resource	Acquire Resources	DINT[0]	equipment ID
Upload the output parameters configured for automatic upload on terminal state or transfer of control	Upload Output Parameters Subset	DINT[0]	terminal = 1 transfer of control = 2
Send a message to an operator	Send Message to Operator	DINT[0]	message ID

If you want to:	Then configure the PXRQ instruction as follows:		
Clear a message from an operator	Clear Message to Operator	DINT[0]	0
Acquire a resource	Acquire Resources	DINT[0]	equipment ID
Acquire multiple resources	Acquire Resources	DINT[0]	equipment ID
		DINT[1]	equipment ID
			...
Release a single resource	Release Resources	DINT[0]	equipment ID
Release multiple resources	Release Resources	DINT[0]	equipment ID
		DINT[1]	equipment ID
			...
Release all resources	Release Resources	DINT[0]	0
Send a message (and optional data) to another phase	Send Message to Linked Phase	DINT[0]	message ID
		DINT[1]	Number of phases to receive message
		DINT[2]	Message Value
		DINT[3]	Message Value
			...
Send a message (and optional data) to another phase and wait for the phase to receive the message	Send Message to Linked Phase and Wait	DINT[0]	message ID
		DINT[1]	Number of phases to receive message
		DINT[2]	Message Value
		DINT[3]	Message Value
			...
Wait to receive a message from another phase	Receive Message From Linked Phase	DINT[0]	message ID

If you want to:	Then configure the PXRQ instruction as follows:		
		DINT[1]	Message Value
		DINT[2]	Message Value
			...
Cancel a message to another phase	Cancel Message to Linked Phase	DINT[0]	message ID
Cancel all messages to another phase	Cancel Message to Linked Phase	DINT[0]	0
Download customer's batch ID	Download Batch Data	DINT[0]	1
		DINT[1]	parameter ID in which to store the value
Download unique batch ID	Download Batch Data	DINT[0]	2
		DINT[1]	parameter ID in which to store the value
Download phase ID	Download Batch Data	DINT[0]	3
		DINT[1]	parameter ID in which to store the value
Download recipe control versus manual phase control	Download Batch Data	DINT[0]	4
		DINT[1]	parameter ID in which to store the value
Download current mode of the phase	Download Batch Data	DINT[0]	5
		DINT[1]	parameter ID in which to store the value
Download the low limit of an input parameter	Download Batch Data	DINT[0]	6 The input parameter tag stores the low limit.
Download the high limit of an input parameter	Download Batch Data	DINT[0]	7 The input parameter tag stores the high limit.
Download the high limit of an input parameter	Download Batch Data	DINT[0]	7

If you want to:	Then configure the PXRQ instruction as follows:		
			The input parameter tag stores the high limit.
Download data about the container currently in use	Download Material Manager Data Container In Use	DINT[0]	1
		DINT[1]	Attribute ID (specify the single attribute)
		DINT[2]	Phase parameter ID (specify the parameter tag to download the value to)
Download data about the current material inside the container currently in use	Download Material Manager Data Container In Use	DINT[0]	2
		DINT[1]	Attribute ID (specify the single attribute)
		DINT[2]	Phase parameter ID (specify the parameter tag to which to download the value)
Download data about the current lot inside the container currently in use	Download Material Manager Data Container In Use	DINT[0]	3
		DINT[1]	Attribute ID (specify the single attribute)
		DINT[2]	Phase parameter ID (specify the parameter tag to which to download the value)
Upload data about the container currently in use	Upload Material Manager Data Container In Use	DINT[0]	1
		DINT[1]	Attribute ID (specify the single attribute)
		DINT[2]	Phase parameter ID (specify the parameter tag from which to upload the value)
Upload data about the current material inside the container currently in use	Upload Material Manager Data Container In Use	DINT[0]	2

If you want to:	Then configure the PXRQ instruction as follows:		
		DINT[1]	Attribute ID (specify the single attribute)
		DINT[2]	Phase parameter ID (specify the parameter tag from which to upload the value)
Upload data about the current lot inside the container currently in use	Upload Material Manager Data Container In Use	DINT[0]	3
		DINT[1]	Attribute ID (specify the single attribute)
		DINT[2]	Phase parameter ID (specify the parameter)
Download the current binding's container priority	Download Container Binding Priority	DINT[0]	parameter ID in which to store the value
Upload a new container priority for the current binding	Upload Container Binding Priority	DINT[0]	parameter ID that has value
Download information regarding the availability of sufficient material	Download Sufficient Material	DINT[0]	parameter ID in which to store the value In the result value: • 0 = insufficient material • 1 = sufficient material
Generate a signature	Generate E Signature	DINT[0]	ID of the signature template
		DINT[1]	
Download material attribute	Download Material Manager Database Data	DINT[0]	0
		DINT[1]	Phase parameter ID (specify the parameter tag to which to download the value)
		DINT[2]	Material controller ID
		DINT[3]	Attribute ID (specify the single attribute)

If you want to:	Then configure the PXRQ instruction as follows:		
Download lot attribute	Download Material Manager Database Data	DINT[0]	1
		DINT[1]	Phase parameter ID (specify the parameter tag to which to download the value)
		DINT[2]	Material controller ID
		DINT[3]	Attribute ID (specify the single attribute)
Download container attribute	Download Material Manager Database Data	DINT[0]	3
		DINT[1]	Phase parameter ID (specify the parameter tag to which to download the value)
		DINT[2]	Material controller ID
		DINT[3]	Attribute ID (specify the single attribute)
Download container priority assignment	Download Material Manager Database Data	DINT[0]	4
		DINT[1]	Phase parameter ID (specify the parameter tag to which to download the value)
		DINT[2]	Material controller ID
		DINT[3]	Attribute ID (specify the single attribute)
		DINT[4]	
Upload material attribute	Upload Material Manager Database Data	DINT[0]	5
		DINT[1]	Phase report ID (specify the phase report tag from which to upload)
		DINT[2]	Material controller ID

If you want to:	Then configure the PXRQ instruction as follows:		
		DINT[3]	Attribute ID (specify the single attribute)
Upload lot attribute	Upload Material Manager Database Data	DINT[0]	6
		DINT[1]	Phase report ID (specify the phase report tag from which to upload)
		DINT[2]	Material controller ID
		DINT[3]	Attribute ID (specify the single attribute)
Upload container attribute	Upload Material Manager Database Data	DINT[0]	8
		DINT[1]	Phase report ID (specify the phase report tag from which to upload)
		DINT[2]	Material controller ID
		DINT[3]	Attribute ID (specify the single attribute)
Download container priority assignment	Download Material Manager Database Data	DINT[0]	9
		DINT[1]	Phase parameter ID (specify the parameter tag to which to download the value)
		DINT[2]	Material controller ID
		DINT[3]	Attribute ID (specify the single attribute)
		DINT[4]	

PXRQ Error Codes

ERR (hex)	EXERR (hex)	Description	Recommended Action
00	0000	The PXRQ instruction was aborted before it sent the request to FactoryTalk Batch software.	None

ERR (hex)	EXERR (hex)	Description	Recommended Action
		The PXRQ instruction sent to Sequence was aborted for one of these reasons: 1. Power cycle. 2. Controller mode change from Run/Test to Program mode.	
01	0000	The PXRQ instruction was aborted after it sent the request to FactoryTalk Batch software. The PXRQ instruction sent to Sequence was aborted because the abort set before PXRQ completed.	None
02	0000	Two or more PXRQ instructions executed at the same time using the same request type.	Limit execution to one PXRQ instruction at a time.
03	0110	Communication error. The request was not delivered because there is no subscriber to the phase.	Check that FactoryTalk Batch software is connected and running.
	0210	Communication error. The request was not delivered because there is no connection to the Notify object.	
	0410	Communication error. Delivery failed.	
	1020	FactoryTalk Batch software is not attached to the phase.	
	2020	Communication error. This indicates that the subscription is delete pending.	
04	0002	The FactoryTalk Batch software encountered an error while processing the request.	Check the connection and communication path to FactoryTalk Batch software.

ERR (hex)	EXERR (hex)	Description	Recommended Action
	0003	The PXRQ instruction contains an invalid value.	
	0004	FactoryTalk Batch software is not in the proper state to process the request.	
	0005	Two or more PXRQ instructions executed at the same time using different request types.	
	0006	Error storing to parameter tags at end of request processing.	
05	0000	FactoryTalk Batch software received the request but passed back an invalid cookie.	Check the connection and communication path to FactoryTalk Batch software.
06	0001	PXRQ sent an invalid request type/parameter to External Sequencer: Acquire Resources passed too many data values (>=100).	
	002	PXRQ sent an invalid request type/parameter to External Sequencer: Download Batch Data needs two parameters data values.	
	0003	PXRQ sent an invalid request type/parameter to External Sequencer: Download Batch Data, Download Material Track Data Container In Use: Batch Parameter out of range (0< <8).	
	0004	PXRQ sent an invalid request type/parameter to External Sequencer: Parameter Id out of range (0<= <=99).	

ERR (hex)	EXERR (hex)	Description	Recommended Action
	0005	PXRQ sent an invalid request type/parameter to External Sequencer: Request type is not supported.	
07	0000	Communication lost while PXRQ is waiting for response from the external sequencer.	Check the connection and communication path to FactoryTalk Batch software.
08	0000	The Execution State of the Equipment Phase associated Sequence Step is not Connected.	
09	0000	Un-supported request type send to Sequence.	
0A	0000	Equipment Phase has no sequencing Input parameters defined.	
0B	0000	Equipment Phase has no sequencing Output parameters defined.	
0C	0000	Equipment Phase is overridden.	
0D	0000	Sequence is not attached to Equipment Phase.	
0E	0000	At least one of Sequence Step Input tag value is invalid.	

Affects Math Status Flags

No

Major/Minor Faults

None. See *Index Through Arrays* for operand-related faults.

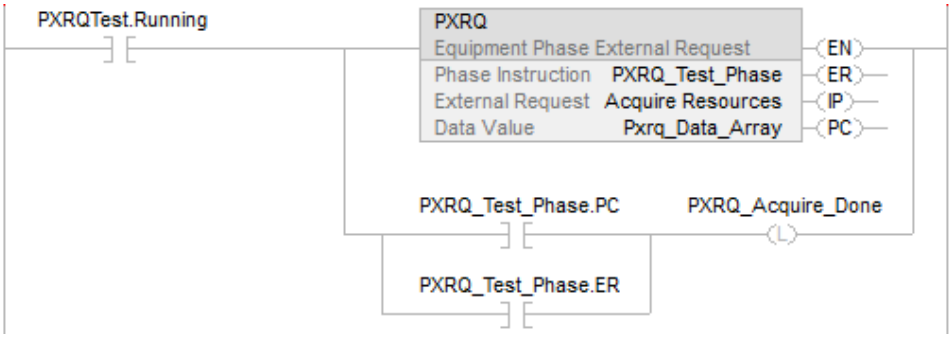
Execution

Condition/State	Action Taken
Prescan	No action taken.

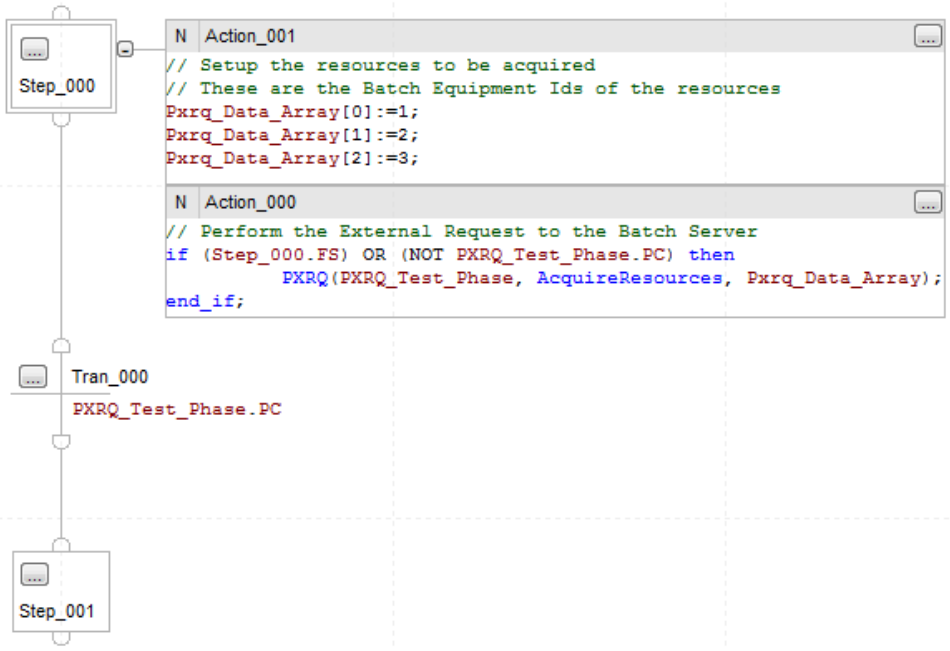
Condition/State	Action Taken
Postscan	No action taken.
EnableIn is false	No action taken.
EnableIn is true	The instruction executes.

Example

Ladder Diagram



Structured Text



Phase Failure (PFL)

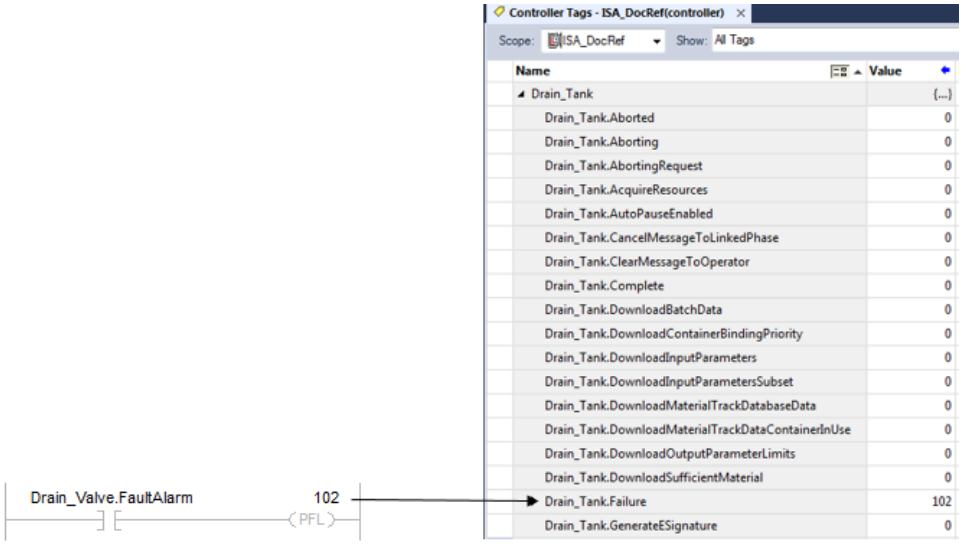
This instruction applies to the Compact GuardLogix 5370, Compact GuardLogix 5380, CompactLogix 5370, CompactLogix 5380, GuardLogix 5570, GuardLogix 5580, ControlLogix 5570, ControlLogix 5580, and ControlLogix 5590 controllers.

When using this instruction with ControlLogix redundancy system, outputs controlled by this instruction may not be bumpless during redundancy switchover, if the instruction and phase program are not scheduled in the highest priority task.

Use the PFL instruction as an optional method to signal a failure for an equipment phase. The PFL instruction sets the value of the failure code for an equipment phase. Use the instruction to signal a specific failure for an equipment phase, such as a specific device has faulted. The PFL instruction sets the failure code only to a value greater than its current value.

This is a transitional instruction. Follow these steps when using it:

- In ladder logic, insert an instruction to toggle the rung-condition-in from false to true each time the instruction should execute.
- In a Structured Text routine, insert a condition for the instruction to cause it to execute only on a transition.



Available Languages

Ladder Diagram

Function Block

This instruction is not available in function block.

Structured Text

PFL (Failure_Code);

Operands

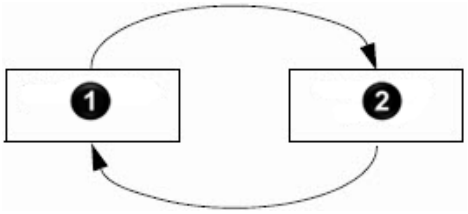
Ladder Diagram

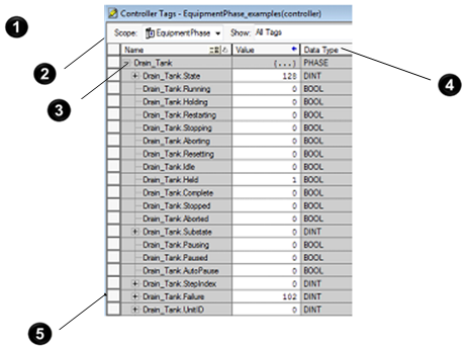
Operand	Type	Format	Description
Failure_Code	DINT	immediate tag	Value to set the failure code for the equipment phase. If a negative failure code given, it evaluates as 0.

Structured Text

The operands are the same as those for the Ladder Diagram PFL instruction.

Guidelines for using the PFL Instruction

Guideline	Details
Put the PFL instruction in the equipment phase.	The PFL instruction sets the failure code for the equipment phase in which the instruction is put. There is *no* operand to identify a specific equipment phase. Typically, put the PFL instruction in a prestate routine for the equipment phase. • The controller always scans the prestate routine, even when an equipment phase is in the idle state. • The controller scans the prestate routine before *each* scan of a state.  **Number** / **Description** 1 — Prestate routine 2 — Current state routine Use the progress routine to continuously monitor the health of an equipment phase as it progresses through its states.

Guideline	Details
Prioritize failure codes.	The PFL instruction sets the failure code only to a value greater than its current value. • For example, if a PFL instruction sets the failure code = 102, another PFL instruction can only set the failure code > 102. • Make sure to assign higher values to exceptions that require higher priority in their handling. Otherwise, a lower priority exception may overwrite a more critical exception.
To take action when a failure occurs, monitor the Failure member of the PHASE tag.	The PFL instruction writes its value to the Failure member of the PHASE tag for the equipment phase.  **Number** / **Description** 1 — When creating an equipment phase, the Logix Designer application creates a tag for the status of the equipment phase. 2 — controller scope 3 — Name = *phase_name* 4 — PHASE data type 5 — The PFL instruction writes its value to the failure member for the equipment phase.

Guideline	Details
To clear the failure code, use a PCLF instruction.	Use a PCLF instruction to clear the failure code of an equipment phase. Instructions such as a CLR or MOV *will not* change the failure code.

Affects Math Status Flags

No.

Major/Minor Faults

A major fault will occur if:	Fault type	Fault code
Instruction is called from outside an Equipment Phase program.	4	91

See *Index Through Arrays* below for array-index faults.

Execution

For Structured Text, EnableIn is always true during normal scan. Therefore, if the instruction is in the control path activated by the logic, it will execute. All conditions below the thick solid line can only occur during Normal Scan mode.

Condition/State	Action Taken
Prescan	No action taken.
Postscan	No action taken.
EnableIn is false	No action taken.
EnableIn is true	The instruction executes as described above.

Example

Ladder Diagram

In the prestate routine of an equipment phase...

If the *Drain_Valve.FaultAlarm* = 1 (The valve did not go to the commander state.) then

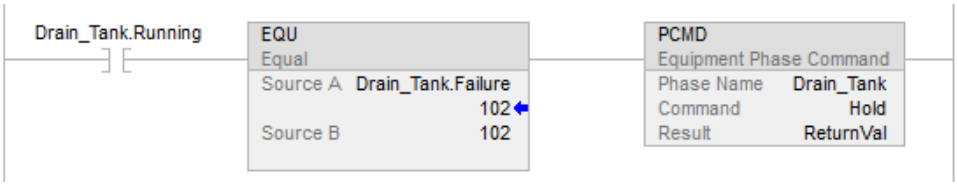
Failure code for the equipment phase = 102.

If *Drain_Tank.Running* = 1 (The *Drain_Tank* equipment phase is in the running state.)

And *Drain_Tank.Failure* = 102 (failure code for the equipment phase)

Then

Change the state of the *Drain_Tank* equipment phase to holding via the hold command.



Structured Text

In the prestate routine of an equipment phase...

```
(*If the drain valve does not go to the commanded state, then set the
failure code of this equipment phase = 102.*)
If Drain_Valve.FaultAlarm Then
    PFL(102);
End_If;

(*If the Drain_Tank equipment phase = running and its failure code = 102,
issue the hold command and send the equipment phase to the holding state.*)
If Drain_Tank.Running And (Drain_Tank.Failure = 102) Then
    PCMD(Drain_Tank,hold,0);
End_IF;
```

Phase New Parameters (PRNP)

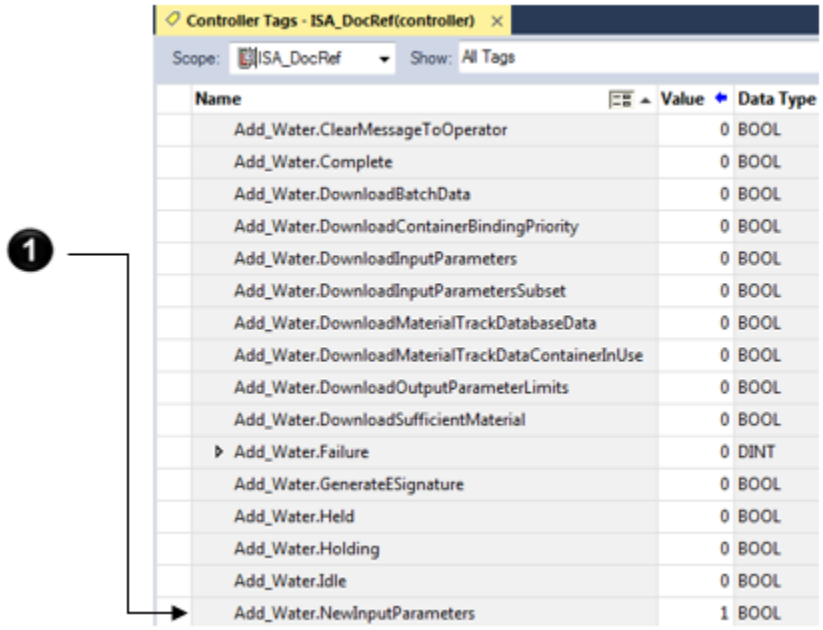
This instruction applies to the Compact GuardLogix 5370, Compact GuardLogix 5380, CompactLogix 5370, CompactLogix 5380, GuardLogix 5570, GuardLogix 5580, ControlLogix 5570, ControlLogix 5580, and ControlLogix 5590 controllers.

When using this instruction with ControlLogix redundancy system, outputs controlled by this instruction may not be bumpless during redundancy switchover, if the instruction and phase program are not scheduled in the highest priority task.

Use the PRNP instruction to clear the NewInputParameters bit of an equipment phase. The PRNP instruction clears the NewInputParameters bit of the equipment phase.

This is a transitional instruction. Follow these steps when using it:

- In ladder logic, insert an instruction to toggle the rung-condition-in from false to true each time the instruction should execute.
- In a Structured Text routine, insert a condition for the instruction to cause it to execute only on a transition.



When FactoryTalk Batch software has new parameters for an equipment phase, it sets the NewInputParameters bit for the phase.After downloading the parameters, use the PRNP instruction to clear the bit.

Available Languages

Ladder Diagram

Function Block

This instruction is not available in function block.

Structured Text

PRNP ( );

Operands

Ladder Diagram

None

Structured Text

None

Enter the parentheses ( ) after the instruction mnemonic, even though there are no operands.

Affects Math Status Flags

No

Major/Minor Faults

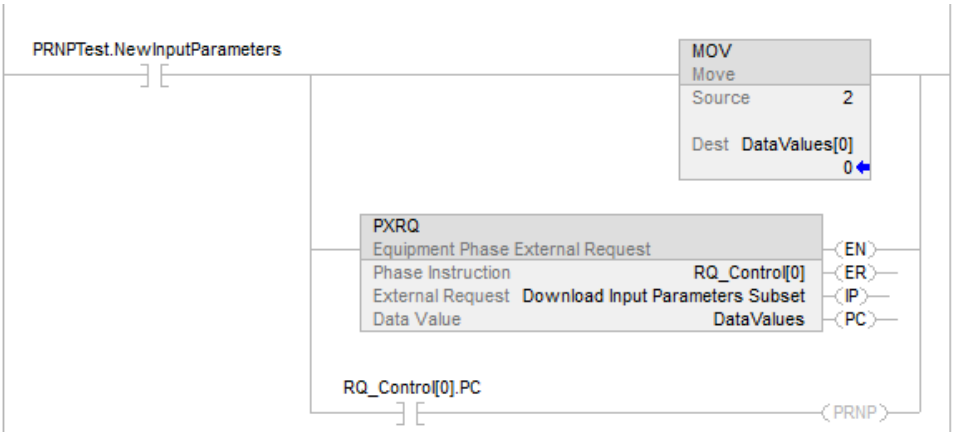
None. See *Index Through Arrays* for operand-related faults.

Execution

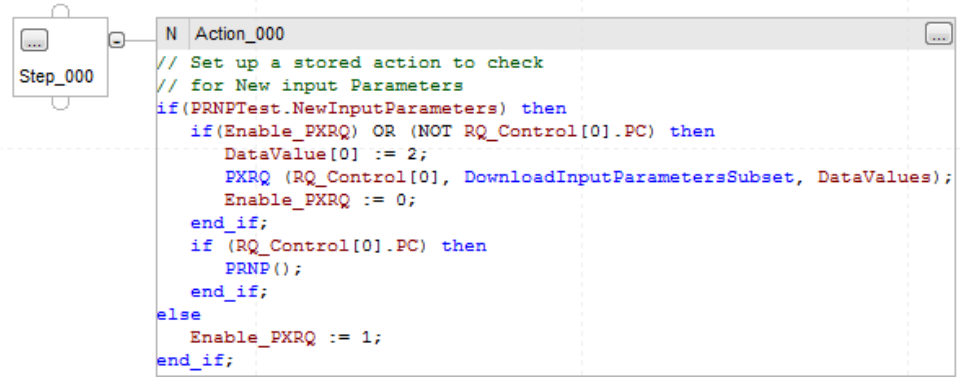
Condition/State	Action Taken
Prescan	No action taken.
Postscan	No action taken.
EnableIn is false	No action taken.
EnableIn is true	The instruction executes.

Example

Ladder Diagram



Structured Text



If PRNPTest.NewInputParameters = 1 (FactoryTalk Batch software has new input parameter for the equipment phase), then

If Enable_PXRQ = 1 (Let the PXRQ instruction execute.)

Or RQ_Control[0].PC = 0 (The PXRQ instruction is in process.), then

Datavalues[0] = this set the PXRQ instruction for transfer of control.

Send the Download Input Parameters Subset request to FactoryTalk Batch software.

Send DataValues[0] = 2, the instruction is set for transfer of control.

Enable_PXRQ = 0 (Do no let the PXRQ instruction restart after the request completes)

If RQ_Control[0].PC = 1 (The request is complete.), then

ThisPhase.NewInputParameters = 0 via the PRNP instruction

Otherwise

Enable_PXRQ = 1 (Let the PXRQ instruction execute the next time new input parameters are available.)

Phase Override Command (POVR)

This instruction applies to the Compact GuardLogix 5370, Compact GuardLogix 5380, CompactLogix 5370, CompactLogix 5380, GuardLogix 5570, GuardLogix 5580, ControlLogix 5570, ControlLogix 5580, and ControlLogix 5590 controllers.
Use the POVR instruction to give a Hold, Stop, or Abort command to an equipment phase, regardless of ownership.

The POVR instruction:

- Gives the Hold, Stop, or Abort command to an equipment phase.
- Overrides all owners of the equipment phase. The command works even if Logix Designer software, HMI, FactoryTalk Batch software, or another program already owns the equipment phase. This instruction does not change the ownership of the equipment phase.
- High priority HMI ownership is specific only to CompactLogix 5370 and ControlLogix 5570 controllers.

When using this instruction with ControlLogix redundancy system, outputs controlled by this instruction may not be bumpless during redundancy switchover, if the instruction and phase program are not scheduled in the highest priority task.

This is a transitional instruction. Follow these steps when using it:

- In ladder logic, insert an instruction to toggle the rung-condition-in from false to true each time the instruction should execute.
- In a Structured Text routine, insert a condition for the instruction to cause it to execute only on a transition.

Available Languages

Ladder Diagram

Function Block

This instruction is not available in function block.

Structured Text

POVR (PhaseName, Command, Result);

Operands

Ladder Diagram

Operand	Type	Format	Description
Phase Name	phase	Name of the equipment phase	Equipment phase to change to a different state
Command	command	Name of the command	One of these commands for the equipment phase: • Hold • Stop • Abort
Result	DINT	immediate tag	To let the instruction return a code for its success or failure, enter a DINT tag in which to store the result code. Otherwise, enter 0.

Structured Text

The operands are the same as those for the Ladder Diagram POVR instruction.

Guidelines for using the POVR Instruction

Guideline	Details
If want to override other owners.	Want the equipment to hold, stop, or abort even if have manual control of the equipment phase via Logix Designer software? • Yes - Use the POVR instruction • No - Use the PCMD instruction This also applies to HMI, FactoryTalk Batch software or other programs. Use the POVR only to hold, stop, or abort, regardless of ownership. For example, suppose the equipment checks for jammed material. If there is a jam, always abort the equipment. In that case, use the POVR instruction. This way, the

Guideline	Details
	equipment aborts even under manual control via Logix Designer software.
Limit execution of a POVR instruction to a single scan.	Limit the execution of the POVR instruction to a single scan. Each command applies to only a specific state or states. Once the equipment phase changes state, the command is *no longer* valid. To limit execution, use methods such as: • Execute the POVR instruction within a P1 Pulse (Rising Edge) or P0 Pulse (Falling Edge) action. • Place a one shot instruction before the POVR instruction. • Execute the POVR instruction and then advance to the next step.

POVR Result Codes

If assigning a tag to store the result of a POVR instruction, the instruction returns one of these codes when it executes:

Code (Dec)	Description
0	Successful command.
24577	Invalid command.
24578	Invalid command for the current state of the equipment phase. For example, if the equipment phase is in the stopping state, then a hold command is not valid.
24594	Unscheduled or inhibited equipment phase or in an inhibited task.

Affects Math Status Flags

No

Major/Minor Faults

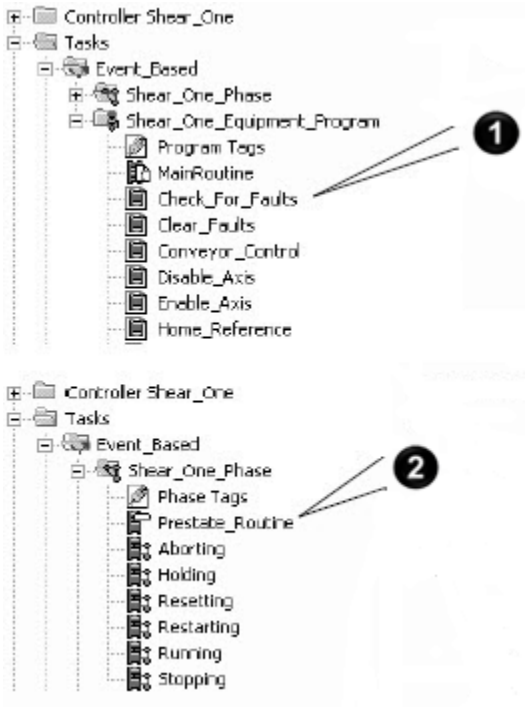
None. See *Index Through Arrays* for operand-related faults.

Execution

Condition/State	Action Taken
Prescan	No action taken.
Postscan	No action taken.
EnableIn is false	No action taken.

Condition/State	Action Taken
EnableIn is true	The instruction executes.

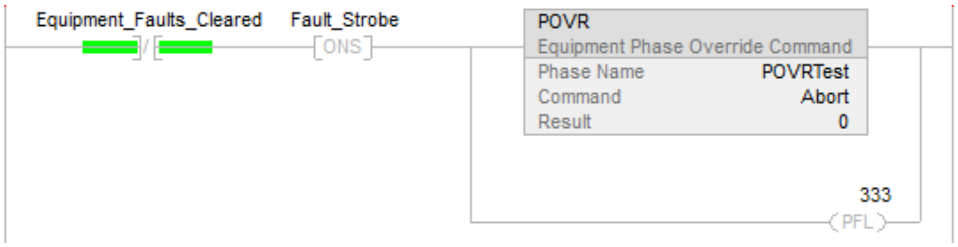
Example



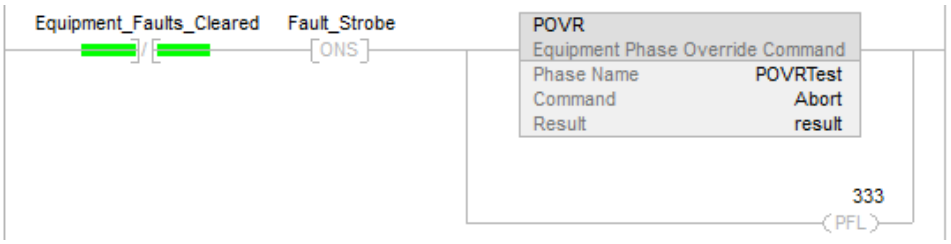
Number	Description
1	The equipment program watches for the these faults: • Faulted axis • Jammed material If there is a fault, then *Local_Interface.Equipment_Faults_Cleared* = 0. This tag is an alias for the controller-scoped tag *Shear_1*.
2	The prestate routine of the equipment phase watches for the equipment program to signal a fault. • If *Interface_To_Equipment.Equipment_Faults_Cleared*=0 then there is a fault. • Both *Interface_To_Equipment* and *Local_Interface* as aliases for *Shear_1*, so they have the same values. If there is a fault, then Give the *Shear_One_Phase* equipment phase the abort command. The POVR instruction makes sure the command works, even if someone has manual control of the equipment phase through Logix Designer software.

Number	Description
	The PFL instruction sets the failure code for *Shear_One_Phase = 333*. The *Fault_Strobe* keeps these actions to a single scan.

Ladder Diagram



Example 2



Structured Text

```
If NOT Equipment_Faults_Cleared And NOT Fault_Strobe then
POVR(POVRTest,Abort, 0);
PFL(333);
end_if;
Fault_Strobe := NOT Equipment_Faults_Cleared;
```

Phase Paused (PPD)

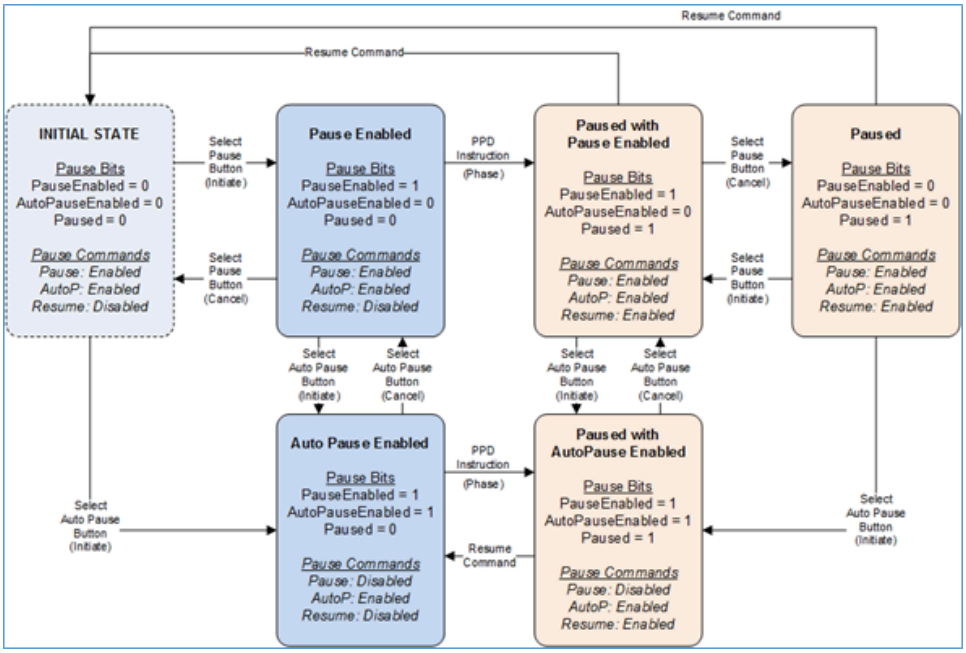
This instruction applies to the Compact GuardLogix 5370, Compact GuardLogix 5380, CompactLogix 5370, CompactLogix 5380, GuardLogix 5570, GuardLogix 5580, ControlLogix 5570, ControlLogix 5580, and ControlLogix 5590 controllers.

When using this instruction with ControlLogix redundancy system, outputs controlled by this instruction may not be bumpless during redundancy switchover, if the instruction and phase program are not scheduled in the highest priority task.

Use the PPD instruction to set up breakpoints within the logic of an equipment phase. Pausing an equipment phase requires configuring break points by coding a PPD instruction in a phase's state routine logic. The phase state routine pauses when the PPD instruction executes and the phase receives a command to pause at its next opportunity.

This is a transitional instruction. Follow these steps when using it:

- In ladder logic, insert an instruction to toggle the rung-condition-in from false to true each time the instruction should execute.
- In a Structured Text routine, insert a condition for the instruction to cause it to execute only on a transition.



Three operator commands relate to the pause functionality:

- **Pause**: The Pause command enables or disables pausing execution of the phase when the next PPD instruction executes. The Pause command toggles the PauseEnabled bit ON (1) or OFF (0).
- **AutoPause:** The AutoPause command enables or disables automatically enabling pausing the phase after processing a Resume command. The AutoPause command toggles the AutoPauseEnabled bit ON (1) or OFF (0).
- **Resume:** The Resume command directs the firmware to resume executing the phase state routine logic. Resume sets the PauseEnabled and Paused bits OFF (0).

The Pause substate uses these three bits:

- **PauseEnabled**: The PauseEnabled bit maintains the status of processing a Pause command. This is bit 0 of the Pause substate.

  When Paused is ON (1), execution of a PPD instruction pauses execution of the state routine's logic. This bit updates when it receives a Pause command (toggling the bit's value). Additionally, a Resume command sets PauseEnabled OFF (0), if the AutoPauseEnabled bit is ON (1).
- **AutoPauseEnabled**: The AutoPauseEnabled bit maintains the status of automatically enabling pausing immediately after a Resume command. This is bit 2 of the Pause substate.

This bit updates when it receives an AutoPause command (toggling the bit's value). When AutoPauseEnabled is ON (1) and the phase is Paused, the Resume command leaves PauseEnabled ON (1).

- **Paused**: The Paused bit maintains the pause-state of the phase, Paused (1) or not-Paused (0). This is bit 1 of the Pause substate. The Paused bit also disables the rest of the rung (RLL), it does not terminate or suspend the execution of the routine.

  This bit is only set by the phase's firmware. When the PauseEnabled bit is ON, execution of a PPD instruction causes the Paused bit to be set to Paused (1) and firmware pauses execution of the phase state routine suspends. A Resume command sets the Paused bit to not-Paused (0) and the phase executes its logic.

Available Languages

Ladder Diagram

Function Block

This instruction is not available in function block.

Structured Text

PPD( );

Operands

Ladder Diagram

None

Structured Text

None

Enter the parentheses ( ) after the instruction mnemonic, even though there are no operands.

Guidelines for using the PPD Instruction

Guideline	Details
Organize logic as a series of steps.	PPD instructions (breakpoints) are easiest to use if the logic moves through defined steps, such as a state machine or SFC. • A breakpoint *only* signals that specific conditions are met. It *does not* stop the execution of the equipment phase. • To have logic actually break (pause) at a breakpoint, organize the logic so that it stays at the step at which the breakpoint occurred until it receives the resume command.

<table>
<tr><th>Guideline</th><th colspan="2">Details</th></tr>
<tr><td>Do not use a PPD instruction as a temporary end of the routine.</td><td colspan="2">Even when an equipment phase pauses, it continues to execute all its logic.
When a PPD instruction executes, it only sets the Paused bit for the equipment phase.
If programing the PPD instruction in RLL, it disables only the rest of the logic on its rung. It does not terminate or suspend the execution of the routine.
Think of the PPD instruction as a condition that applies or is ignored based on the auto pause and pause commands.</td></tr>
<tr><td>Use PPD instruction to pause at the same breakpoint over several scans.</td><td colspan="2">When the PauseEnabled bit is TRUE, an equipment phase goes to Paused at the first PPD instruction whose conditions are true. If the PPD instruction executes over several scans, the equipment phase may continually pause at the same breakpoint.</td></tr>
<tr><td>Make sure only 1 PPD instruction at a time is true.</td><td colspan="2">A PPD instruction does not have a control tag to remember whether it executed.
Anytime its conditions are true (and the equipment phase is in a substate with PauseEnabled True), the PPD instruction acts as a breakpoint (and pauses the phase by disabling the rest of the logic on the rung).
Limiting logic to one possible breakpoint at a time ensures a pause at the required breakpoint.</td></tr>
<tr><td>Choose the substate to use.</td><td colspan="2">PPD instructions (breakpoints) work only when the equipment phase PauseEnabled bit is True.<table><tr><td>To pause at:</td><td>Give this command:</td></tr><tr><td>Each true breakpoint</td><td>Auto Pause</td></tr><tr><td>First true breakpoint</td><td>Pause</td></tr></table></td></tr>
</table>

Affects Math Status Flags

No

Major/Minor Faults

A major fault occurs if:	Fault type	Fault code
Instruction is called from outside an Equipment Phase program.	4	91

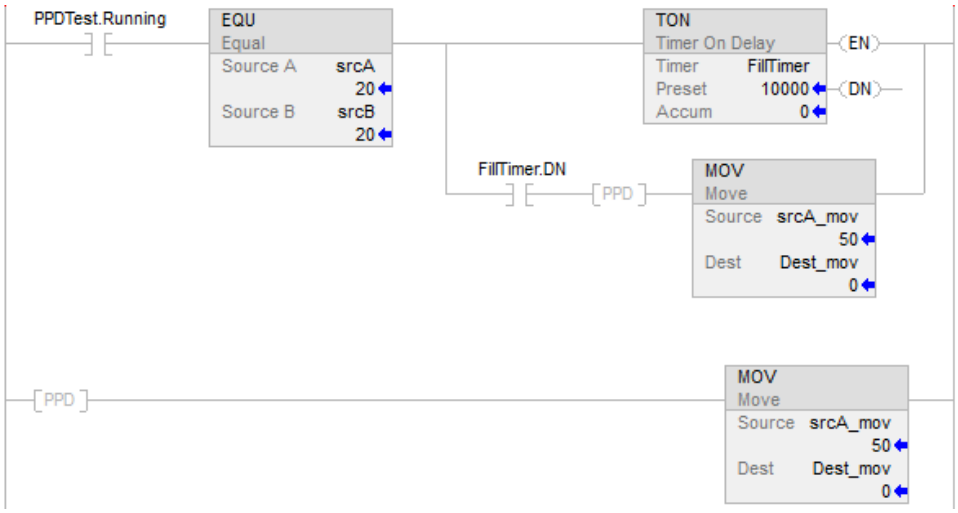
If an Add-On Instruction uses a PPD instruction, and a non-equipment phase program calls the Add-On Instruction, Logix Designer gives a warning. Check the Add-On Instruction for this instruction to disallow it. See *Index Through Arrays* for operand-related faults.

Execution

Condition/State	Action Taken
Prescan	No action taken.
Postscan	No action taken.
EnableIn is false	No action taken.
EnableIn is true	The instruction executes.

Example

Ladder Diagram



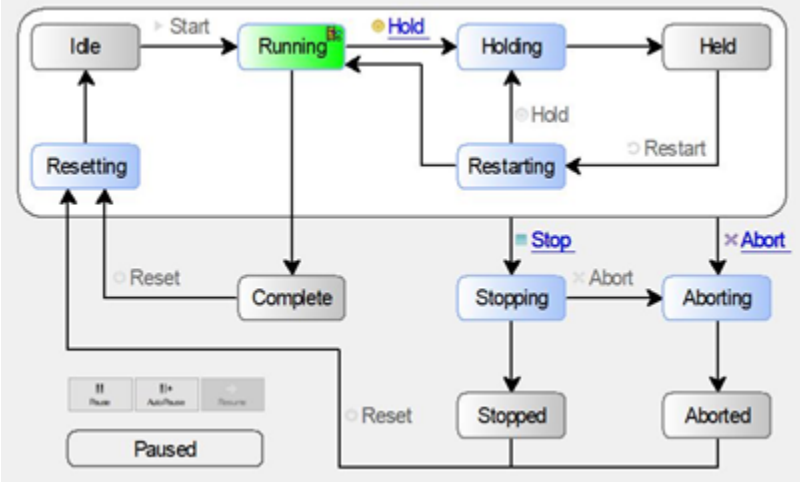
Phase State Complete (PSC)

This instruction applies to the Compact GuardLogix 5370, Compact GuardLogix 5380, CompactLogix 5370, CompactLogix 5380, GuardLogix 5570, GuardLogix 5580, ControlLogix 5570, ControlLogix 5580, and ControlLogix 5590 controllers.

When using this instruction with ControlLogix redundancy system, outputs controlled by this instruction may not be bumpless during redundancy switchover, if the instruction and phase program are not scheduled in the highest priority task.

Use the PSC instruction to signal an equipment phase that the state routine is complete to indicate that it should go to the next state.

The PSC instruction signals the completion of a phase state routine.



In the running state routine, use the PSC instruction to transition the equipment phase to the complete state.

- This is a transitional instruction. Follow these steps when using it:
  - In ladder logic, insert an instruction to toggle the rung-condition-in from false to true each time the instruction should execute.
  - In a Structured Text routine, insert a condition for the instruction to cause it to execute only on a transition.

Available Languages

Ladder Diagram

Function Block

This instruction is not available in function block.

Structured Text

PSC( );

Operands

Ladder Diagram

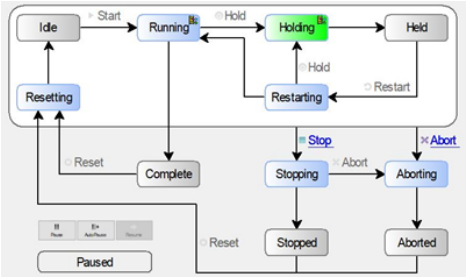
None

Structured Text

None

Enter the parentheses ( ) after the instruction mnemonic, even though there are no operands.

Guidelines for using the PSC Instruction

Guideline	Details
Use the PSC instruction in *each* phase state routine that is added to an equipment phase.	Without a PSC instruction, the equipment phase remains in the state and does *not* go to the next state. • Place the PSC instruction as the last step in the phase state routine. • When the state is done (complete), execute the PSC instruction.
	In the holding state routine, use the PSC instruction to let the equipment phase go to the Held state
Remember that the PSC instruction does *not* stop the current scan of a routine.	When the PSC instruction executes, the controller scans the rest of the routine and then transitions the equipment phase to the next state. The PSC instruction does not terminate the execution of the routine.
Do *not* use a PSC instruction in a prestate routine.	Use the PSC instruction only to signal the transition from one state to another.

Affects Math Status Flags

No

Major/Minor Faults

A major fault will occur if:	Fault type	Fault code
Instruction is called from outside an equipment phase program.	4	91

If an Add-On Instruction used a PSC instruction and a non-equipment phase program calls the Add-On Instruction, Logix Designer gives a warning. Check the Add-On Instruction for this instruction and disallow it. See *Index Through Arrays* for operand-related faults.

Execution

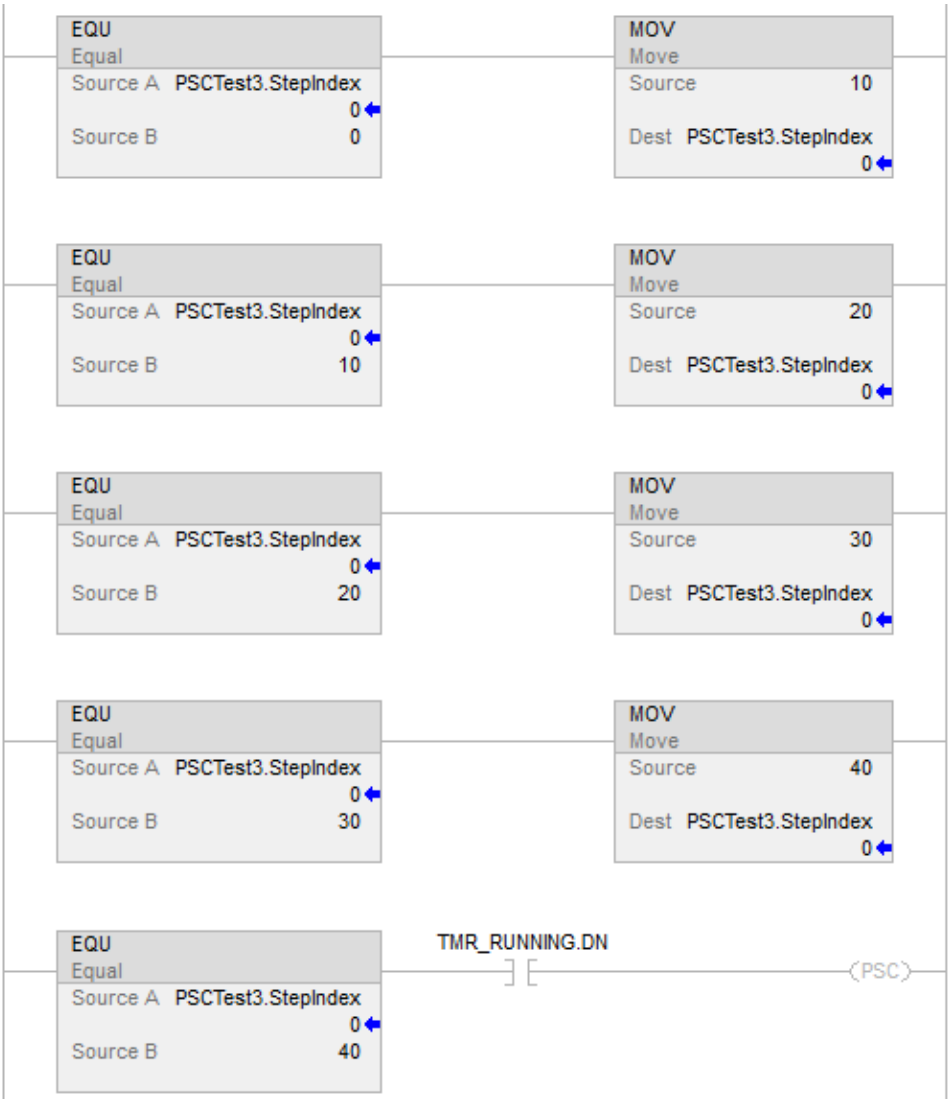
In structured text, instructions execute each time they are scanned. To limit the scan of an instruction, use a qualifier of an SFC action, a structured text construct, or both.

Condition/State	Action Taken
Prescan	No action taken.

Condition/State	Action Taken
Postscan	No action taken.
EnableIn is false	No action taken.
EnableIn is true	The instruction executes.

Examples

Ladder Diagram



Structured Text

```
If TagEnableRunning
And PSCTest.Running Then
PSC();
End_if;
```

Equipment Sequence instructions

The table lists the command instructions for Equipment Sequences. The instructions are available for routines that use the Ladder Diagram and the Structured Text programming languages. They are not available for use in routines that use the Function Block and the Sequential Function Chart programming languages.

Instruction	Description	Structured Text format
Attach to Equipment Sequence (SATT)	Request that the parent program of the user routine be the owner of the Equipment Sequence.	SequenceName, Result
Detach from Equipment Sequence (SDET)	Release ownership of an Equipment Sequence.	SequenceName
Equipment Sequence Command (SCMD)	Send a command to the Equipment Sequence.	SequenceName, Command, Result
Equipment Sequence Clear Failure (SCLF)	Clear the failure code of an Equipment Sequence.	SequenceName
Equipment Sequence Override (SOVR)	Send a HOLD, STOP, or ABORT command to an Equipment Sequence, regardless of ownership.	SequenceName, Command, Result
Equipment Sequence Assign Sequence Identifier (SASI)	Assign an ID string to the Equipment Sequence.	SequenceName, Sequence ID, Result

Attach to Equipment Sequence (SATT)

This instruction applies to the Logix Designer 5580P controllers and to ControlLogix 5590P controllers. Logix Designer 5580P controllers and ControlLogix 5590P controllers also support controller redundancy.

When using this instruction with ControlLogix redundancy system, outputs controlled by this instruction may not be bumpless during redundancy switchover, if the instruction and sequence program are not scheduled in the highest priority task.

WARNING:
When using redundancy with an Equipment Sequence, sequence execution may not be as expected after switchover if the phase and sequence are not scheduled on the same task.

Use the Attach to Equipment Sequence (SATT) instruction to take ownership of an Equipment Sequence. An Equipment Sequence may be commanded by a program if the program either owns the Equipment Sequence or the Equipment Sequence has no owners. A tag must be assigned to store the result code of an SATT instruction.

The SATT instruction returns one of five result codes. Result code 0 indicates that the SATT instruction ran successfully. The other four codes indicate that the instruction did not run successfully and provide additional information about the reason for the instruction failure.

This is a transitional instruction. Follow these steps when using it:

- In ladder logic, insert an instruction to toggle the rung-condition-in from false to true each time the instruction should execute.
- In a Structured Text routine, insert a condition for the instruction to cause it to execute only on a transition.

Available Languages

Ladder Diagram

Function Block Diagram

This instruction is not available in function block.

Structured Text:

SATT(Sequence Name, Result)

Operands

Ladder Diagram

Operand	Data Type	Format	Description
Sequence Name	SEQUENCE	name of the Equipment Sequence	Equipment Sequence that you want to change to own (attach) to command.
Result	DINT	Tag	For the instruction to return a success or failure code, enter a DINT tag where the result code is to be stored. Otherwise, enter 0.

Structured Text

The operands are the same as for the Ladder Diagram.

Guidelines for using the SATT Instruction

Guideline	Details
Remember that the Logix Designer application overrides ownership of the Equipment Sequence.	Regardless of whether a program or FactoryTalk Batch software owns an Equipment Sequence, the option exists to use Logix Designer to override ownership and command the Equipment Sequence.

Guideline	Details
The Equipment Sequence must be either owned by the program to command it or have no owners for the program to command it.	The ownership instructions are **Attach** (SATT) and **Detach** (SDET). Internal Sequencers (programs), external sequencers (FactoryTalk Batch), and operators use an **Attach** command to command an Equipment Sequence.
When the Equipment Sequence is done, relinquish ownership.	To relinquish ownership, use a Detach from Equipment Sequence (SDET) instruction.
Avoid making unnecessary command requests if the Equipment Sequence is generating sequence events.	Unnecessary command requests can flood the event processing buffers and cause you to miss significant events.
Use the **Result** code to verify ownership, and include steps that should take place if the attachment fails because the Equipment Sequence is owned by another program or operator.	Use the **Result** operand to get a code that shows the success or failure of the SATT instruction. On each execution, the SATT instruction tries to take ownership of the Equipment Sequence. When a program or operator owns an Equipment Sequence, another execution of the SATT instruction fails and produces result code 24582. When you use the SATT instruction, either: • Limit its execution to a single scan to avoid the 24582 result code - Include this in your conditions for ownership: **result code = 24582**

SATT Result Codes

Code (Dec)	Description
0	The command was successful.
24579	The Equipment Sequence cannot be commended. Logix Designer already owns the Equipment Sequence. The caller attached to the Equipment Sequence, but it is not the current commanding owner. • This program now also owns the Equipment Sequence. - Since the Logix Designer is a higher priority, the program cannot command the Equipment Sequence.
24582	The program already owns the Equipment Sequence.
24593	One of these already owns the Equipment Sequence. • An external Sequencer such as FactoryTalk Batch software. - Another program in the controller. - An HMI operator

Code (Dec)	Description
24594	The Equipment Sequence is unscheduled, inhibited, or in a task that is inhibited.

Affects Math Status Flags

No

Major/Minor Faults

None specific to this instruction. See Index Through Arrays[1] for operand related faults.

Execution

At instruction execution, the SATT instruction attempts to take ownership of the Equipment Sequence.

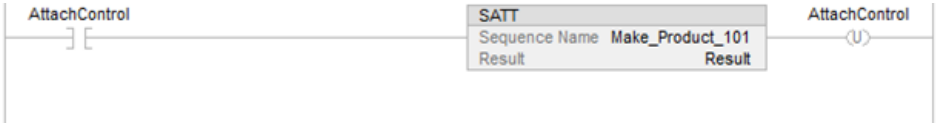
Ladder Diagram

Condition	Action Taken
Prescan	No action taken
Rung-condition-in is false	No action taken
Rung-condition-in is true	• The instruction executes • The rung-condition-out is set to true
Postscan	No action taken

Structured Text

Condition	Action Taken
Prescan	No action taken
Rung-condition-in is false	No action taken
Rung-condition-in is true	The instruction executes
Postscan	No action taken

Example



Structured Text

```
if (AttachControl) then
SATT(Make_Product_101, Result);
end_if
```

Guidelines for SATT instructions

Keep these guidelines in mind when using the Attach to Equipment Sequence (SATT) instruction.

Guideline	Details
Remember that the Logix Designer application overrides ownership of the Equipment Sequence.	Ownership makes sure that a program can command the Equipment Sequence, and it locks out any other sequencers.
The Equipment Sequence must be owned by the program to command it.	Both Equipment Sequences and Equipment Phases must be *owned* to be commanded. The ownership commands are **Attach** (SATT) and **Detach** (SDET). Internal sequencers (programs), external sequencers (FactoryTalk Batch), and operators use an **Attach** instruction to command an Equipment Sequence.
When the sequence is done, relinquish ownership.	To relinquish ownership, use a Detach from Equipment Sequence (SDET) instruction.
Avoid making unnecessary command requests if the equipment sequence is generating sequence events.	Unnecessary command requests can flood the event processing buffers and cause you to miss significant events.
Use the **Result** code to verify ownership, and include steps that should take place if the attachment fails because the Equipment Sequence is owned by another program or by the operator.	Use the **Result** operand to get a code that shows the success or failure of the SATT instruction. On each execution, the SATT instruction tries to take ownership of the Equipment Sequence. When a program or operator owns an Equipment Sequence, another execution of the SATT instruction fails and produces result code 24582. When you use the SATT instruction, either: • Limit its execution to a single scan to avoid the 24582 result code. • Include the following in your conditions for ownership: result code = 24582.

Result codes for SATT instructions

When a tag is assigned to store the result of an Attach to Equipment Sequence (SATT) instruction, the instruction returns one of these codes when it runs.

Code (Dec)	Description

0	The command was successful.
24579	The Equipment Sequence cannot be commanded for these reason: • The program successfully attached to the Equipment Sequence, but it cannot command the sequence because Logix Designer, a higher priority application, has overridden ownership.
24582	The program already owns the Equipment Sequence.
24593	One of these already owns the equipment phase. • An external sequencer such as FactoryTalk Batch software. • Another program in the controller.
24594	The Equipment Sequence is unscheduled, inhibited, or in a task that is inhibited.

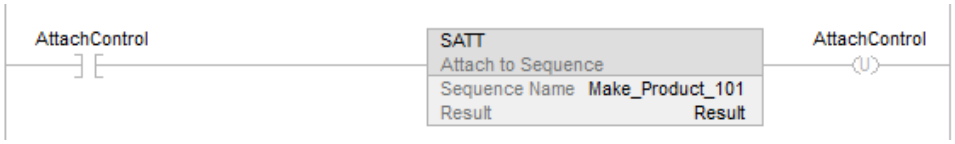
Use the **Result** operand to get a code that shows the success or failure of the SATT instruction. The **Result** operand should contain either **0** or a DINT tag, depending on whether ownership conflicts or other errors are likely to occur.

- If ownership conflicts or other errors are not likely, enter **0** in the **Result** operand.
- If ownership conflicts or other errors are likely, enter a DINT tag in the **Result** operand. The DINT tag stores a code for the result of the execution of the instruction.

SATT instruction examples

The examples show the SATT instruction as it appears in a Ladder Diagram and in Structured Text.

Ladder Diagram



Structured Text

```
if (AttachControl) then
SATT (Make_Product_101, Result);
end_if;
```

Detach from Equipment Sequence (SDET)

This instruction applies to the Logix Designer 5580P controllers and to ControlLogix 5590P controllers. Logix Designer 5580P controllers and ControlLogix 5590P controllers also support controller redundancy.

When using this instruction with ControlLogix redundancy system, outputs controlled by this instruction may not be bumpless during redundancy switchover, if the instruction and sequence program are not scheduled in the highest priority task.

WARNING:
When using redundancy with an Equipment Sequence, sequence execution may not be as expected after switchover if the phase and sequence are not scheduled on the same task.

Use the Detach from Equipment Sequence (SDET) instruction to relinquish ownership of an Equipment Sequence. After a program executes an SDET instruction, the program no longer owns the Equipment Sequence. The Equipment Sequence is then available for ownership by another program or by FactoryTalk Batch software. Use the SDET instruction only if the program previously took ownership of an Equipment Sequence through an Attach to Equipment Sequence (SATT) instruction.

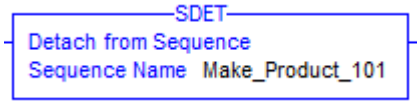
This is a transitional instruction. Follow these steps when using it:

- In ladder logic, insert an instruction to toggle the rung-condition-in from false to true each time the instruction should execute.
- In a Structured Text routine, insert a condition for the instruction to cause it to execute only on a transition.

Available Languages

The SDET instruction is available in these languages.

- Ladder diagram



- Structured text: SDET(SequenceName)

Supported Operands

The SDET instruction uses this operand.

Operand	Type	Format	Description
Sequence Name	Sequence	name of the Equipment Sequence	Equipment Sequence for which you want to relinquish ownership.

Arithmetic status flags and fault conditions

Arithmetic status flags are not affected by the SDET instruction. The SDET instruction cannot trigger a fault, so there are no fault conditions for this instruction.

Instruction execution

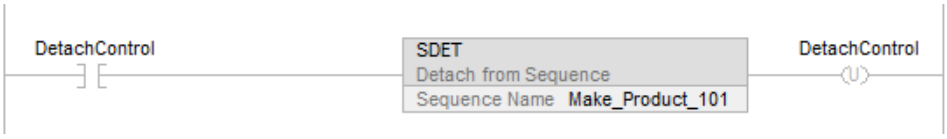
The table describes the execution steps for SDET instructions.

Condition	Ladder Diagram Action	Structured Text Action
Prescan	The rung-condition-out is set to false.	No action taken.
Rung-condition-in is false	The rung-condition-out is set to false.	N/A
Rung-condition-in is true	• The instruction executes. • The rung-condition-out is set to true.	N/A
Scan of structured text	N/A	In structured text, instructions execute each time they are scanned. To limit the scan of an instruction, use a qualifier of an SFC action or a structured text construct that includes a condition, such as if, then, or else.
Instruction execution	The instruction relinquishes ownership of the specified Equipment Sequence.	The instruction relinquishes ownership of the specified Equipment Sequence.
Postscan	The rung-condition-out is set to false.	No action taken.

SDET instruction examples

The following examples show the SDET instruction as it appears in a ladder diagram and in structured text.

Ladder Diagram



Structured Text

```
if (DetachControl) then
SDET (Make_Product_101);
end_if;
```

Equipment Sequence Assign Sequence Identifier (SASI)

This instruction applies to the Logix Designer 5580P controllers and to ControlLogix 5590P controllers. Logix Designer 5580P controllers and ControlLogix 5590P controllers also support controller redundancy.

When using this instruction with ControlLogix redundancy system, outputs controlled by this instruction may not be bumpless during redundancy switchover, if the instruction and sequence program are not scheduled in the highest priority task.

WARNING:
When using redundancy with an Equipment Sequence, sequence execution may not be as expected after switchover if the phase and sequence are not scheduled on the same task.

Use the Assign Sequence Identifier (SASI) instruction to assign a sequence ID to the Equipment Sequence. You can only set the sequence ID when these prerequisites are met:

- The controller is online.
- The Equipment Sequence is in the IDLE state.
- You have taken ownership of the Equipment Sequence, or there is no other owner of the Equipment Sequence.

The sequence ID can be up to 82 characters in length, using these printable ASCII characters:

a-z, A-Z, 0-9, !"#$%&'()*+,-./:;<=>?@[\] ^_`{|}~ and space

This is a transitional instruction. Follow these steps when using it:

- In ladder logic, insert an instruction to toggle the rung-condition-in from false to true each time the instruction should execute.
- In a Structured Text routine, insert a condition for the instruction to cause it to execute only on a transition.

When a tag is assigned to store the result of an SASI instruction, the instruction returns a result code when it runs. Result code 0 indicates that the SASI instruction ran successfully. The other codes indicate that the instruction did not run successfully and provide additional information about the reason for the instruction failure.

Available Languages

Ladder Diagram

Function Block Diagram

This instruction is not available in Function Block.

Structured Text

SASI(Sequence Name, Sequence Id, Result)

Operands

Ladder Diagram

Operand	Data Type	Format	Description
Sequence Name	SEQUENCE	name of the Equipment Sequence	Equipment Sequence to which you want to assign an identifier.
Sequence Id	STRING	Tag	Enter a STRING tag in which the identifier is stored, or a quoted string containing up to 82 characters.
Result	DINT	Tag	For an instruction to return a success or failure code, enter a DINT tag where the result code is to be stored. Otherwise, enter 0.

Structured Text

The operands are the same as for the Ladder Diagram.

SASI Result Codes

Code (Dec)	Description
0	the Sequence Id was successfully assigned.
24578	The Sequence state is not IDLE or there presently is a Sequence failure.
24579	Sequence is attached by another owner.

Affects Math Status Flags

No

Major/Minor Faults

None specific to this instruction. See Index Through Arrays[1] for operand related faults.

Execution

At instruction execution, the SASI instruction attempts to assign a string identifier to the specified Equipment Sequence.

Ladder Diagram

Condition	Action Taken
Prescan	No action taken
Rung-condition-in is false	No action taken
Rung-condition-in is true	The instruction executes
Postscan	No action taken

Structured Text

Condition	Action Taken
Prescan	No action taken
Rung-condition-in is false	No action taken
Rung-condition-in is true	The instruction executes
Postscan	No action taken

Example

Ladder Diagram

The Sequence ID parameter can be a STRING tag in which the identifier is stored, or a quoted string containing up to 82 characters.

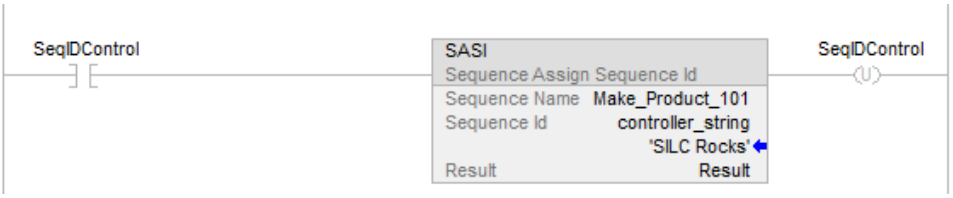
Structured Text

```
if (SasiControl) then
SASI(Make_Product_101, IdString, Result);
end_if
```

SASI instruction examples

The examples show the SASI instruction as it appears in a ladder diagram and in structured text.

Ladder Diagram Example



The Sequence ID parameter can be a STRING tag in which the identifier is stored, or a quoted string containing up to 82 characters.

Structured Text Example

```
if (IdControl) then
SASI (Make_Product_101, NewId, Results);
end_if;
```

Equipment Sequence Diagram instructions

The table describes the Equipment Sequence Diagram instructions.

Icon	Icon Name	Description
	Add Step and Transition Pair	Use **Add Step and Transition Pair** to add a step and transition pair. Although added as a pair, you can select and edit each element separately.
	Add Disconnected Step	Use the **Add Disconnected Step** to add a step without adding a transition.
	Add Disconnected Transition	Use **Add Disconnected Transition** to add a transition without adding a step.
	Add Simultaneous Divergence	Use **Add Simultaneous Divergence** to create a branch where all linked steps execute simultaneously.
	Add Selective Divergence	Use **Add Selective Divergence** to create a divergence for a selective branch. In a selective divergence, only one of multiple paths is executed--the path containing the transition that first evaluates as TRUE.

	Add Simultaneous Convergence	Use **Add Simultaneous Convergence** to merge simultaneous execution paths back together.
	Add Selective Convergence	Use **Add Selective Convergence**  to merge selective divergent paths back into one execution path in the selective branch.

Equipment Sequence Clear Failure (SCLF)

This instruction applies to the Logix Designer 5580P controllers and to ControlLogix 5590P controllers. Logix Designer 5580P controllers and ControlLogix 5590P controllers also support controller redundancy.

When using this instruction with ControlLogix redundancy system, outputs controlled by this instruction may not be bumpless during redundancy switchover, if the instruction and sequence program are not scheduled in the highest priority task.

WARNING:
When using redundancy with an Equipment Sequence, sequence execution may not be as expected after switchover if the phase and sequence are not scheduled on the same task.

Use the Equipment Sequence Clear Failure (SCLF) instruction to clear the failure code of an Equipment Sequence. Keep this in mind when using the SCLF instruction.

- A CLR instruction, MOV instruction, or assignment does not change the failure code of an Equipment Sequence.
- The Equipment Sequence cannot have other owners when you use the SCLF instruction. The SCLF instruction does not clear the failure code if the Logix Designer application, FactoryTalk Batch software, or another program owns the Equipment Sequence.
- An Equipment Sequence refuses a RESUME command until it is cleared of failures.

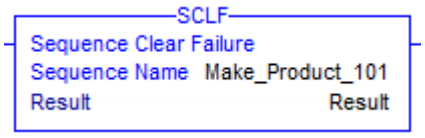
This is a transitional instruction. Follow these steps when using it:

- In ladder logic, insert an instruction to toggle the rung-condition-in from false to true each time the instruction should execute.
- In a Structured Text routine, insert a condition for the instruction to cause it to execute only on a transition.

Available Languages

The SCLF instruction is available in these languages.

- Ladder diagram



- Structured text: SCLF(SequenceName)

Supported Operands

The SCLF instruction uses these operands.

Operand	Type	Format	Description
Sequence Name	Sequence	name of the Equipment Sequence	Equipment Sequence for which you want to clear a failure code.
Result	DINT	Immediate tag	For an instruction to return a success or failure code, enter a DINT tag where the result code is stored. Otherwise, enter **0**.

Arithmetic status flags and fault conditions

Arithmetic status flags are not affected by the SCLF instruction. The SCLF instruction cannot trigger a fault, so there are no fault conditions for this instruction.

Instruction execution

The table describes the execution steps for SCLF instructions.

Condition	Ladder Diagram Action	Structured Text Action
Prescan	The rung-condition-out is set to false.	No action taken.
Rung-condition-in is false	The rung-condition-out is set to false.	N/A
Rung-condition-in is true	• The instruction executes. • The rung-condition-out is set to true.	N/A
Scan of structured text	N/A	In structured text, instructions execute each time they are scanned. To limit the scan of an instruction, use a qualifier of an SFC action or

		a structured text construct that includes a condition, such as if, then, or else.
Instruction execution	The instruction clears the value of the failure code for the specified Equipment Sequence.	The instruction clears the value of the failure code for the specified Equipment Sequence.
Postscan	The rung-condition-out is set to false.	No action taken.

Equipment Sequence command (SCMD)

This instruction applies to the Logix Designer 5580P controllers and to ControlLogix 5590P controllers. Logix Designer 5580P controllers and ControlLogix 5590P controllers also support controller redundancy.

When using this instruction with ControlLogix redundancy system, outputs controlled by this instruction may not be bumpless during redundancy switchover, if the instruction and sequence program are not scheduled in the highest priority task.

WARNING:
When using redundancy with an Equipment Sequence, sequence execution may not be as expected after switchover if the phase and sequence are not scheduled on the same task.

Use the Equipment Sequence command (SCMD) instruction to change the state of an Equipment Sequence. The SCMD instruction can send these commands to an Equipment Sequence: START, RESTART, HOLD, STOP, ABORT, RESET, PAUSE, RESUME, and AUTOPAUSE. The calling program must either be attached as owner to the Equipment Sequence or there is no owner of the Equipment Sequence before the SCMD instruction can run. Use the SATT instruction to attach to an Equipment Sequence. In addition, the Equipment Sequence must be in the correct state (see chart below) for the command to execute successfully.

Like the SCMD instruction, the Equipment Sequence Override instruction (SOVR) also changes the state of an Equipment Sequence, but it changes the state regardless of ownership. If the SCMD instruction must execute regardless of ownership, use an SOVR instruction instead of an SCMD instruction.

IMPORTANT: The SOVR instruction is intended for emergencies only. Control Engineers should use caution when deciding to use it.

When a tag is assigned to store the result of an SCMD instruction, the instruction returns one of five result codes when it runs. Result code 0 indicates that the SCMD instruction ran successfully. The other four codes indicate that the instruction did not run successfully and provide additional information about the reason for the instruction failure.

This is a transitional instruction. Follow these steps when using it:

- In ladder logic, insert an instruction to toggle the rung-condition-in from false to true each time the instruction should execute.
- In a Structured Text routine, insert a condition for the instruction to cause it to execute only on a transition.

Valid command states for the SCMD instruction

The SCMD instruction command transitions an Equipment Sequence to another state. SCMD instruction commands may only be processed in certain states, with the exceptions of PAUSE, RESUME, and AUTOPAUSE, which are valid in all states. The table lists the states in which commands are valid.

Command	Valid in these states
START	Valid in the IDLE state.
RESTART	Valid in the HELD state.
HOLD	Valid in the RUNNING and RESTARTING states.
STOP	Valid in the RUNNING, HOLDING, RESTARTING, and HELD states.
ABORT	Valid in the RUNNING, HOLDING, RESTARTING, STOPPING, and HELD states.
RESET	Valid in the ABORTED, STOPPED, and COMPLETE states.

Available Languages

Ladder Diagram

Function Block

This instruction is not available in function block.

Structured Text

SCMD(Sequence Name, Sequence Command, Result)

Operands

Ladder Diagram

Operand	Data Type	Format	Description
Sequence Name	SEQUENCE	Name of the Equipment Sequence	Equipment Sequence to perform the command.
Command	Command Enum	Enumeration of the command	Command to send to the Equipment Sequence. Send one of these commands: START, RESTART, HOLD, STOP, ABORT, RESET, PAUSE, RESUME, or AUTOPAUSE.
Result	DINT	Tag	For an instruction to return a success or failure code, enter a DINT tag where the result code is stored. Otherwise, enter 0.

Structured Text

The operands are the same as for the Ladder Diagram.

Guidelines for using the SCMD Instruction

Guideline	**Details**
Limit execution of the SCMD instruction to a single scan.	Limit the execution of the SCMD instruction to a single scan. Each command applies to only a specific state or states. Once the Equipment Sequence changes state, the command is no longer valid. To limit execution, use methods such as: Run the SCMD instruction within a P1 Pulse (Rising Edge) or P0 Pulse (Falling Edge) action. Place a one-shot instruction before the SCMD instruction. Run the SCMD instruction and then advance to the next step.
The Equipment Sequence must be either owned by the program to command it or have no owners for the program to command it.	The ownership instructions are Attach (SATT) and Detach (SDET).

SCMD Result Codes

Code (Desc)	Description
0	The command was successful.
24578	The command is not valid for the current state of the Sequence.
24579	The caller is attached to this Sequence, but it is not the current Owner of it. A higher priority application is currently the Owner of this sequence.
24580	The caller is not attached to the Sequence.
24594	The Equipment Sequence is unscheduled, inhibited, or in a task that is inhibited.
24604	An equal or higher priority command is being processed.
24631	Too many sequence parameter or step tags are defined per step, so events cannot be handled and the START command failed.

Arithmetic Math Status Flags

No

Major/Minor Faults

None specific to this instruction. See Index Through Arrays for operand related faults.

Execution

At instruction execution, the SCMD instruction commands the specified Equipment Sequence.

Ladder Diagram

Condition	Action Taken
Prescan	No action taken
Rung-condition-in is false	No action taken
Rung-condition-in is true	The instruction executes.
Postscan	No action taken

Structured Text

Condition	Action Taken
Prescan	No action taken

Condition	Action Taken
Rung-condition-in is false	No action taken
Rung-condition-in is true	The instruction executes.
Postscan	No action taken

Example

Ladder Diagram

Structured Text

```
if (CommandControl) then
SCMD(Make_Product_101, Start, Result);
end_if
```

Equipment Sequence Override (SOVR)

This instruction applies to the Logix Designer 5580P controllers and to ControlLogix 5590P controllers. Logix Designer 5580P controllers and ControlLogix 5590P controllers also support controller redundancy.

When using this instruction with ControlLogix redundancy system, outputs controlled by this instruction may not be bumpless during redundancy switchover, if the instruction and sequence program are not scheduled in the highest priority task.

WARNING:
When using redundancy with an Equipment Sequence, sequence execution may not be as expected after switchover if the phase and sequence are not scheduled on the same task.

Use the Equipment Sequence Override (SOVR) instruction to send a HOLD, STOP, or ABORT command to an Equipment Sequence, regardless of ownership.

IMPORTANT: The SOVR instruction is intended for emergencies only. Control Engineers should use caution when deciding to use it.

When a tag is assigned to store the result of an SOVR instruction, the instruction returns one of five result codes when it runs. Result code **0** indicates that the SOVR instruction ran successfully. The other four codes indicate that the instruction did not run successfully and provide additional information about the reason for the instruction failure.

This is a transitional instruction. Follow these steps when using it:

- In ladder logic, insert an instruction to toggle the rung-condition-in from false to true each time the instruction should execute.
- In a Structured Text routine, insert a condition for the instruction to cause it to execute only on a transition.

Available Languages

Ladder Diagram

Function Block

This instruction is not available in function block.

Structured Text

SOVR(SequenceName, Sequence Command, Result)

Operands

Ladder Diagram

Operand	Data Type	Format	Description
Sequence Name	SEQUENCE	Name of the Equipment Sequence	Equipment Sequence to perform the command.
Command	Command Enum	Name of the command	Command to send to the Equipment Sequence. Send one of these commands: HOLD, STOP, or ABORT
Result	DINT	Tag	For an instruction to return a success or failure code, enter a DINT tag where the result code is to be stored. Otherwise, enter 0.

Structured Text

The operands are the same as the Ladder Diagram.

Guidelines for using the SOVR Instruction

Guideline	Details
Make sure you want to override other owners.	Under most circumstances, use the SCMD instruction to programmatically command an Equipment Sequence. However, use the SOVR instruction to command an Equipment Sequenceunder these conditions: • When you are giving the HOLD, STOP, or ABORT command, and the command must always execute under all ownership circumstances. • If the HOLD, STOP, or ABORT command must execute even when you have manual control of the Equipment Sequence through the Logix Designer application or when another program, such as the FactoryTalk Batch software, owns the Equipment Sequence.
Limit execution of the SOVR instruction to a single scan.	Limit the execution of the SOVR instruction to a single scan. Each command applies to only a specific state or states. Once the Equipment Sequence changes state, the command is no longer valid. To limit execution, use methods such as: • Run the SOVR instruction within a P1 Pulse (Rising Edge) or P0 Pulse (Falling Edge) action. • Place a one-shot instruction before the SOVR instruction. • Run the SOVR instruction and then advance to the next step.
Avoid making unnecessary command requests if the Equipment Sequence is generating sequence events.	Unnecessary command requests can flood the event processing buffers and cause you to miss significant events.

SOVR Result Codes

Code (Desc)	Description
0	The command was successful.
24579	The caller is attached to this Sequence, but it is not the current Owner of it. A higher priority application is currently the Owner of this sequence.
24582	The caller already has override ownership of this Sequence.
24583	This sequence's attachment table is full.

Code (Desc)	Description
24606	The caller has already established an external attachment or override as a different owner type.

Affects Math Status Flags

No

Major/Minor Faults

None specific to this instruction. See Index Through Arrays[1] for operand related faults.

Execution

At instruction execution, the SOVR instruction attempts to command the specified Equipment Sequence.

Ladder Diagram

Condition	Action Taken
Prescan	No action taken
Rung-condition-in is false	No action taken
Rung-condition-in is true	The instruction executes.
Postscan	No action taken

Structured Text

Condition	Action Taken
Prescan	No action taken
Rung-condition-in is false	No action taken
Rung-condition-in is true	The instruction executes
Postscan	No action taken

Example

Ladder Diagram

Structured Text

if (OverrideControl) then

```
SOVR(Make_Product_101, Abort, Result);
end_if
```

Guidelines for SCMD instructions

Keep these guidelines in mind when using the Equipment Sequence Command (SCMD) instruction. The SCMD instruction can send these commands: START, RESTART, HOLD, STOP, ABORT, and RESET.

Guideline	Details
Limit execution of the SCMD instruction to a single scan.	Limit the execution of the SCMD instruction to a single scan. Each command applies to only a specific state or states. Once the Equipment Sequence changes state, the command is no longer valid. To limit execution, use methods such as: • Run the SCMD instruction within a P1 Pulse (Rising Edge) or P0 Pulse (Falling Edge) action. • Place a one-shot instruction before the SCMD instruction. • Run the SCMD instruction and then advance to the next step.
The Equipment Sequence must be owned by the program to command it.	Both Equipment Sequences and Equipment Phases must be *owned* to be commanded. The ownership commands are **Attach** (SATT) and **Detach** (SDET). Internal sequencers (programs), external sequencers (FactoryTalk Batch), and operators use an **Attach** instruction to command an Equipment Sequence.

Guidelines for SOVR instructions

Use the Equipment Sequence Override (SOVR) instruction to send a HOLD, STOP, or ABORT command to an Equipment Sequence, regardless of ownership.

IMPORTANT: The SOVR instruction is intended for emergencies only. Control Engineers should use caution when deciding to use it.

Keep these guidelines in mind when using the Equipment Sequence Override (SOVR) instruction.

Guideline	Details
Make sure you want to override other owners.	Under most circumstances, use the SCMD instruction to programmatically command an Equipment Sequence.

	However, use the SOVR instruction to command an Equipment Sequence under these conditions: • When you are giving the HOLD, STOP, or ABORT command, and the command must always execute under all ownership circumstances. • If the HOLD, STOP, or ABORT command must execute even when you have manual control of the Equipment Sequence through the Logix Designer application or when another program, such as the FactoryTalk Batch software, owns the Equipment Sequence.
Limit execution of the SOVR instruction to a single scan.	Limit the execution of the SOVR instruction to a single scan. Each command applies to only a specific state or states. Once the Equipment Sequence changes state, the command is no longer valid. To limit execution, use methods such as: • Run the SOVR instruction within a P1 Pulse (Rising Edge) or P0 Pulse (Falling Edge) action. • Place a one-shot instruction before the SOVR instruction. • Run the SOVR instruction and then advance to the next step.
Avoid making unnecessary command requests if the Equipment Sequence is generating sequence events.	Unnecessary command requests can flood the event processing buffers and cause you to miss significant events.

Result codes for SCLF instructions

When a tag is assigned to store the result of an Equipment Sequence Clear Failure (SCLF) instruction, the instruction returns one of these codes when it executes.

Code (Dec)	Description
0	The command was successful.
48	The command was not executed because it was not possible at the time to generate an event to record the command. • If the command was an ABORT command, the ABORT command is still executed even if the event could not be generated. • This code only occurs if event generation has been enabled in the **Equipment Sequence Properties - Configuration** tab.

24578	The command is not valid for the current state of the Equipment Sequence. For example, if the Equipment Sequence is stopped, then a stop command is not valid.
24594	The Equipment Sequence is unscheduled, inhibited, or in a task that is inhibited.

Use the **Result** operand to get a code that shows the success or failure of the SCLF instruction. The **Result** operand should contain either **0** or a DINT tag, depending on whether ownership conflicts or other errors are likely to occur.

- If ownership conflicts or other errors are not likely, enter **0** in the **Result** operand.
- If ownership conflicts or other errors are likely, enter a DINT tag in the **Result** operand. The DINT tag stores a code for the result of the execution of the instruction.

Result codes for SCMD instructions

When a tag is assigned to store the result of an Equipment Sequence command (SCMD) instruction, the instruction returns one of these codes when it runs.

Code (Dec)	Description
0	The command was successful.
48	The command was not executed because it was not possible at the time to generate an event to record the command. • If the command was an ABORT command, the ABORT command is still executed even if the event could not be generated. • This code only occurs if event generation has been enabled in the **Equipment Sequence Properties - Configuration** tab.
24577	The command is not valid.
24578	The command is not valid for the current state of the Equipment Sequence. For example, if the Equipment

	Sequence is in the running state, then a start command is not valid.
24579	• The Equipment Sequence cannot be commanded for these reason: - The program successfully attached to the Equipment Sequence, but it cannot command the sequence because Logix Designer, a higher priority application, has overridden ownership.
24582	• Attachment to the Equipment Sequence failed because the sequence was previously attached to one of these users: - An external sequencer, such as FactoryTalk Batch software, has ownership. - Another program in the controller (an internal sequencer) has ownership. - An operator using the Sequence Manager ActiveX Controls has ownership.
24580	The caller of the instruction is attached, but is not the current owner of the Equipment Sequence. A higher priority owner, such as Logix Designer, is commanding the Equipment Sequence.
24594	The Equipment Sequence is unscheduled, inhibited, or in a task that is inhibited.
24604	An equal or higher priority command is being processed.
24631	Too many sequence parameter or step tags are defined per step, so events cannot be handled and the START command failed.

Use the **Result** operand to get a code that shows the success or failure of the SCMD instruction. The **Result** operand should contain either **0** or a DINT tag, depending on whether ownership conflicts or other errors are likely to occur.

- If ownership conflicts or other errors are not likely, enter **0** in the **Result** operand.
- If ownership conflicts or other errors are likely, enter a DINT tag in the **Result** operand. The DINT tag stores a code for the result of the execution of the instruction.

Result codes for SOVR instructions

When a tag is assigned to store the result of an Equipment Sequence Override (SOVR) instruction, the instruction returns one of these codes when it executes.

Code (Dec)	Description
0	The command was successful.

48	The command was not executed because it was not possible at the time to generate an event to record the command. • If the command was an ABORT command, the ABORT command is still executed even if the event could not be generated. • This code only occurs if event generation has been enabled in the **Equipment Sequence Properties - Configuration** tab.
24577	The command is not valid.
24578	The command is not valid for the current state of the Equipment Sequence. For example, if the Equipment Sequence is stopped, then a stop command is not valid.
24594	The Equipment Sequence is unscheduled, inhibited, or in a task that is inhibited.

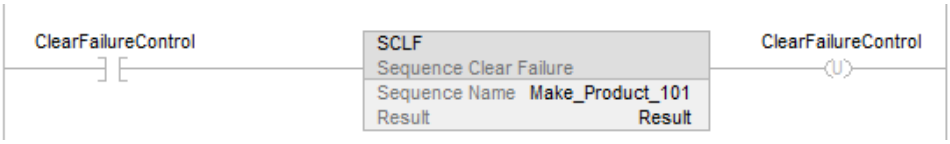
Use the **Result** operand to get a code that shows the success or failure of the SOVR instruction. The **Result** operand should contain either **0** or a DINT tag, depending on whether ownership conflicts or other errors are likely to occur.

- If ownership conflicts or other errors are not likely, enter **0** in the **Result** operand.
- If ownership conflicts or other errors are likely, enter a DINT tag in the **Result** operand. The DINT tag stores a code for the result of the execution of the instruction.

SCLF instruction examples

The examples show the SCLF instruction as it appears in a ladder diagram and in structured text.

Ladder Diagram Example



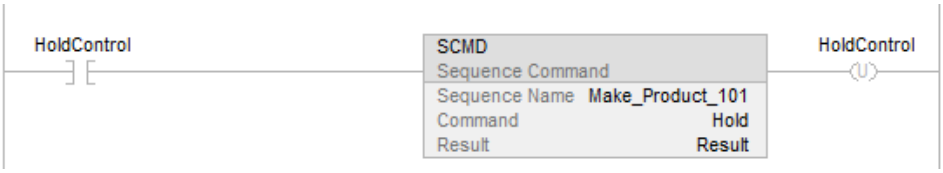
Structured Text Example

```
if (ClearFailureControl) then
SCLF (Make_Product_101);
end_if;
```

SCMD instruction examples

The examples show the Equipment Sequence command (SCMD) instruction as it appears in a Ladder Diagram and in Structured Text.

Ladder Diagram



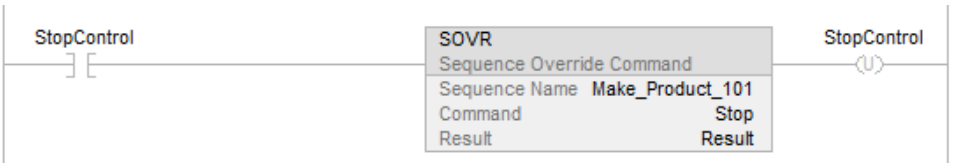
Structured Text

```
if (HoldControl) then
SCMD (Make_Product_101), Hold, Result);
end_if;
```

SOVR instruction examples

The examples show the SOVR instruction as it appears in a Ladder Diagram and in Structured Text.

Ladder Diagram



Structured Text

```
if (StopControl) then
SOVR (Make_Product_101, Stop, Results);
end_if;
```

When should I use an SOVR instruction instead of an SCMD instruction?

- Under most circumstances, use the SCMD instruction to programmatically command an Equipment Sequence. However, use the SOVR instruction to command an Equipment Sequence under these conditions:
  - When you are giving the HOLD, STOP, or ABORT command, and the command must always execute under all ownership circumstances.
  - If the HOLD, STOP, or ABORT command must execute even when you have manual control of the Equipment Sequence through the Logix Designer application or when another program, such as the FactoryTalk Batch software, owns the Equipment Sequence.

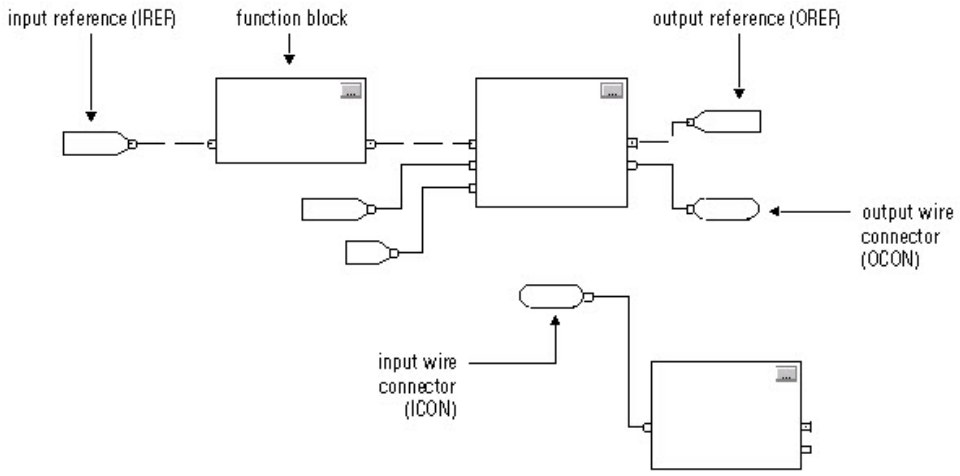
For example, suppose your equipment checks for jammed material. If there is a jam, you always want the equipment to abort. In that case, use the SOVR instruction. This way, the equipment aborts even if you have manual control through the Logix Designer application.

Function Block Attributes

Click a topic below for more information on issues that are unique to function block programming. Review this information to make sure you understand how your function block routines will operate.

Choose the Function Block Elements

To control a device, use these elements:



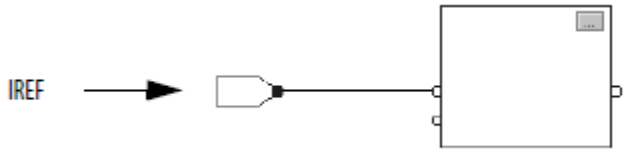
Use the following table to help you choose your function block elements:

If you want to supply a value from an input device or tag	Then use an input reference (IREF)
Send a value to an output device or tag	Output reference (OREF)
Perform an operation on an input value or values and produce an output value or values.	Function block
Transfer data between function blocks when they are: • Far apart on the same sheet • On different sheets within the same routine	Output wire connector (OCON) and an input wire connector (ICON)
Disperse data to several points in the routine	Single output wire connector (OCON) and multiple input wire connectors (ICON)

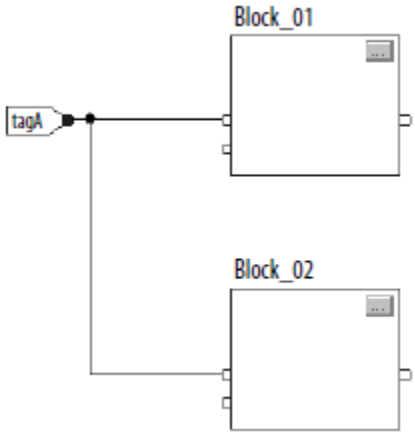
The function block moves the input references into the block structure. If necessary, the function block converts those input references to REAL values. The function block executes and moves the results into the output references. Again, if necessary, the function block converts those result values from REAL to the data types for the output references.

Latching Data

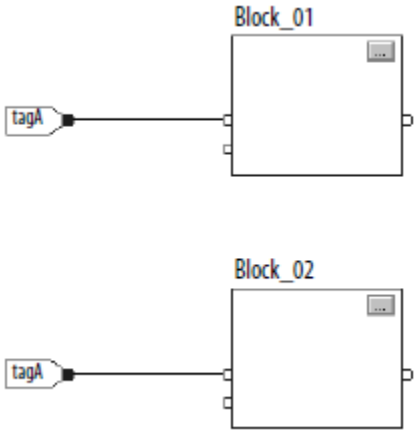
If you use an IREF to specify input data for a function block instruction, the data in that IREF is latched for the scan of the function block routine. The IREF latches data from program-scoped and controller-scoped tags. The controller updates all IREF data at the beginning of each scan.



In this example, the value of tagA is stored at the beginning of the routine's execution. The stored value is used when Block_01 executes. The same stored value is also used when Blcock_02 executes. If the value of tagA changes during execution of the routine, the stored value of tagA in the IREF does not change until the next execution of the routine.

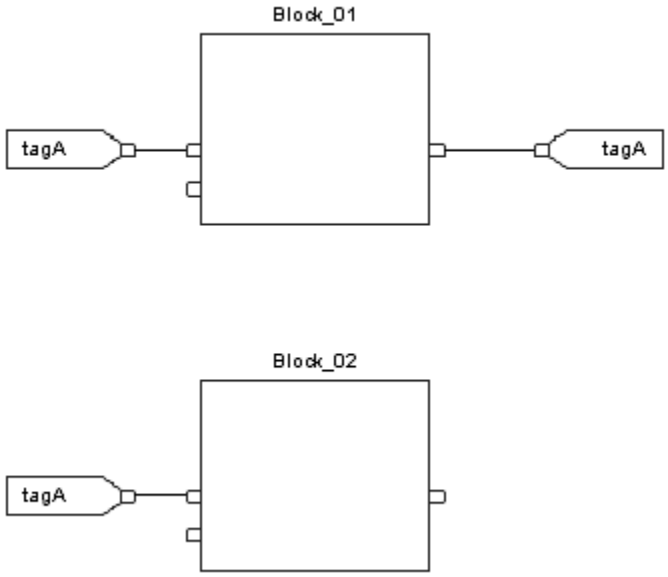


This example is the same as the one above. The value of tagA is stored only once at the beginning of the routine's execution. The routine uses this stored value throughout the routine.



You can use the same tag in multiple IREFs and an OREF in the same routine. Because the values of tags in IREFs are latched every scan through the routine, all IREFs will use the same value, even if an OREF obtains a different tag value during execution of the routine.

In this example, if tagA has a value of 25.4 when the routine starts executing this scan, and Block_01 changes the value of tagA to 50.9, the second IREF wired into Block_02 will still use a value of 25.4 when Block_02 executes this scan. The new tagA value of 50.9 will not be used by any IREFs in this routine until the start of the next scan.



Function Block Responses to Overflow Conditions

In general, the function block instructions that maintain history do not update history with ±NAN, or ±INF values when an overflow occurs. Each instruction has one of these responses to an overflow condition.

Response	Instruction
Response 1 Blocks execute their algorithm and check the result for ±NAN or ±INF. If ±NAN or ±INF, the block outputs ±NAN or ±INF.	ALM NTCH DEDT PMUL DERV POSP ESEL RLIM FGEN RMPS HPF SCRV LDL2 SEL LDLG SNEG LPF SRTP MAVE SSUM MAXC TOT

	MINC UPDN MSTD MUX
Response 2 Blocks with output limiting execute their algorithm and check the result for ±NAN or ±INF. The output limits are defined by the HighLimit and LowLimit input parameters. If ±INF, the block outputs a limited result. If ±NAN, the output limits are not used and the block outputs ±NAN.	HLL, INTG, PI, PIDE, SCL, SOC
Response 3 The overflow condition does not apply. These instructions typically have a boolean output.	BAND, BNOT, BOR, BXOR, CUTD, D2SD, D3SD, DFF, JKFF, OSFI, OSRI, RESD, RTOR, SETD, TOFR, TONR

Order of Execution

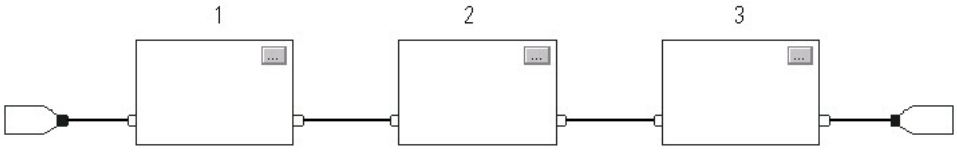
The Logix Designer programming application automatically determines the order of execution for the function blocks in a routine when you:

- verify a function block routine
- verify a project that contains a function block routine
- download a project that contains a function block routine

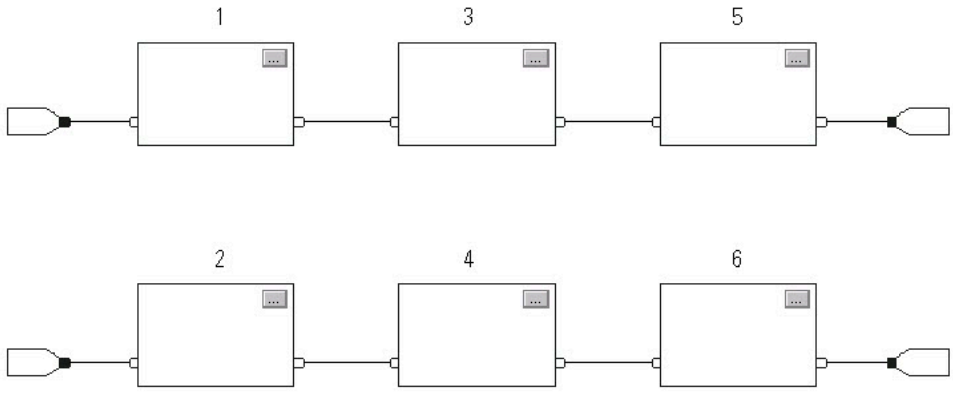
You define execution order by wiring function blocks together and indicating the data flow of any feedback wires, if necessary.

If function blocks are not wired together, it does not matter which block executes first. There is no data flow between the blocks

If you wire the blocks sequentially, the execution order moves from input to output. The inputs of a block require data to be available before the controller can execute that block. For example, block 2 has to execute before block 3 because the outputs of block 2 feed the inputs of block 3.

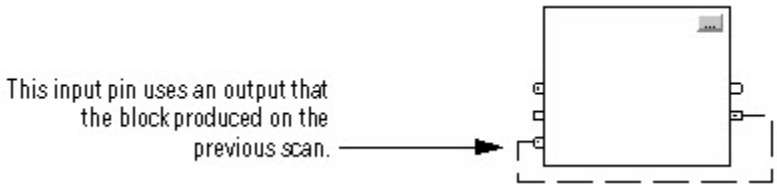


Execution order is only relative to the blocks that are wired together. The following example is fine because the two groups of blocks are not wired together. The blocks within a specific group execute in the appropriate order in relation to the blocks in that group.

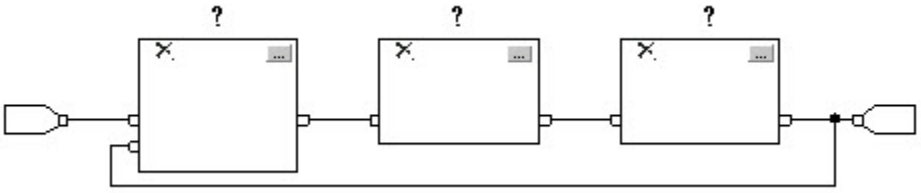


Resolve a Loop

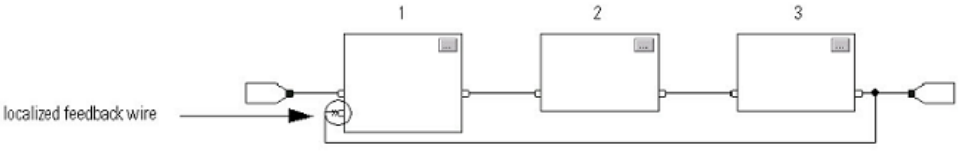
To create a feedback loop around a block, wire an output pin of the block to an input pin of the same block. The following example is OK. The loop contains only a single block, so execution order does not matter.



If a group of blocks are in a loop, the controller cannot determine which block to execute first. In other words, it cannot resolve the loop.

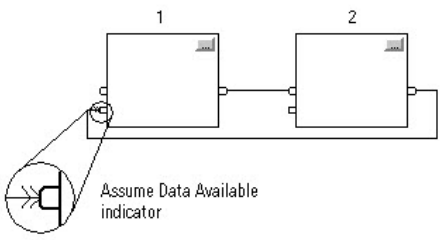
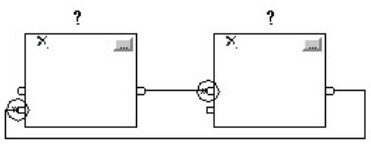


To identify which block to execute first, mark the input wire that creates the loop (the feedback wire) with the *Assume Data Available* indicator. In the following example, block 1 uses the output from block 3 that was produced in the previous execution of the routine.



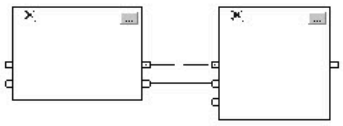
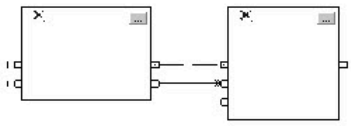
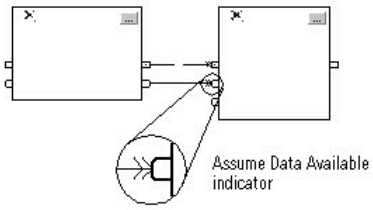
The *Assume Data Available* indicator defines the data flow within the loop. The arrow indicates that the data serves as input to the first block in the loop.

Do *not* mark all the wires of a loop with the *Assume Data Available* indicator.

This is OK	This is NOT OK
 The *Assume Data Available* indicator defines the data flow within the loop.	 The controller cannot resolve the loop because all the wires use the *Assume Data Available* indicator.

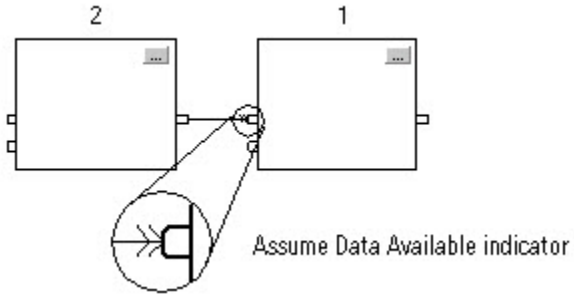
Resolve Data Flow Between Two Blocks

If you use two or more wires to connect two blocks, use the same data flow indicators for all of the wires between the two blocks.

This is OK	This is NOT OK
 Neither wire uses the *Assume Data Available* indicator.	 One wire uses the *Assume Data Available* indicator while the other wire does not.
 Both wires use the *Assume Data Available* indicator.	

Create a One Scan Delay

To produce a one scan delay between blocks, use the Assume Data Available indicator. In the following example, block 1 executes first. It uses the output from block 2 that was produced in the previous scan of the routine.



Summary

In summary, a function block routine executes in this order:

1. The controller latches all data values in IREFs.
2. The controller executes the other function blocks in the order determined by how they are wired.
3. The controller writes outputs in OREFs.

Timing Modes

These process control and drives instructions support different timing modes.

• DEDT	• LDLG	• RLIM
• DERV	• LPF	• SCRV
• HPF	• NTCH	• SOC
• INTG	• PI	• TOT
• LDL2	• PIDE	

There are three different timing modes.

<table>
<tr><th>Timing Mode</th><th colspan="2">Description</th></tr>
<tr><td rowspan="4">Periodic</td><td colspan="2">Periodic mode is the default mode and is suitable for most control applications. We recommend that you place the instructions that use this mode in a routine that executes in a periodic task. The delta time (DeltaT) for the instruction is determined as follows:</td></tr>
<tr><td>If the instruction executes in a</td><td>Then DeltaT equals</td></tr>
<tr><td>Periodic task</td><td>Period of the task</td></tr>
<tr><td>Event or continuous task</td><td>Elapsed time since the previous execution

The controller truncates the elapsed time to whole milliseconds (ms). For example, if the elapsed time = 10.5 ms, the controller sets DeltaT = 10 ms.</td></tr>
<tr><td colspan="2">The update of the process input needs to be synchronized with the execution of the task or sampled 5-10 times faster than the task executes in order to minimize the sampling error between the input and the instruction.</td></tr>
<tr><td>Oversample</td><td colspan="2">In oversample mode, the delta time (DeltaT) used by the instruction is the value written into the OversampleDT parameter of the instruction. If the process input has a time stamp value, use the real time sampling mode instead.</td></tr>
</table>

	Add logic to your program to control when the instruction executes. For example, you can use a timer set to the OversampleDeltaT value to control the execution by using the EnableIn input of the instruction. The process input needs to be sampled 5-10 times faster than the instruction is executed in order to minimize the sampling error between the input and the instruction.
Real time sampling	In the real time sampling mode, the delta time (DeltaT) used by the instruction is the difference between two time stamp values that correspond to the updates of the process input. Use this mode when the process input has a time stamp associated with its updates and you need precise coordination. The time stamp value is read from the tag name entered for the RTSTimeStamp parameter of the instruction. Normally this tag name is a parameter on the input module associated with the process input. The instruction compares the configured RTSTime value (expected update period) against the calculated DeltaT to determine if every update of the process input is being read by the instruction. If DeltaT is not within 1 millisecond of the configuration time, the instruction sets the RTSMissed status bit to indicate that a problem exists reading updates for the input on the module.

Time-based instructions require a constant value for DeltaT in order for the control algorithm to properly calculate the process output. If DeltaT varies, a discontinuity occurs in the process output. The severity of the discontinuity depends on the instruction and range over which DeltaT varies.

A discontinuity occurs if the following happens:

- Instruction is not executed during a scan.
- Instruction is executed multiple times during a task.
- Task is running and the task scan rate or the sample time of the process input changes.
- User changes the time-base mode while the task is running.
- Order parameter is changed on a filter block while the task is running.
- Changing the Order parameter selects a different control algorithm within the instruction.

Common Instruction Parameters for Timing Modes

The instructions that support time-base modes have these input and output parameters.

Input Parameters

Input Parameter	Data Type	Description
TimingMode	DINT	Selects timing execution mode. Value: Description: 0 Periodic mode 1 Oversample mode

		2 Real time sampling mode Valid = 0 to 2 Default = 0 When TimingMode = 0 and task is periodic, periodic timing is enabled and DeltaT is set to the task scan rate. When TimingMode = 0 and task is event or continuous, periodic timing is enabled and DeltaT is set equal to the elapsed time span since the last time the instruction was executed. When TimingMode = 1, oversample timing is enabled and DeltaT is set to the value of the OversampleDT parameter. When TimingMode = 2, real time sampling timing is enabled and DeltaT is the difference between the current and previous time stamp values read from the module associated with the input. If TimingMode invalid, the instruction sets the appropriate bit in Status.
OversampleDT	REAL	Execution time for oversample timing. The value used for DeltaT is in seconds. If TimingMode = 1, then OversampleDT = 0.0 disables the execution of the control algorithm. If invalid, the instruction sets DeltaT = 0.0 and sets the appropriate bit in Status. Valid = 0 to 4194.303 seconds Default = 0.0
RTSTime	DINT	Module update period for real time sampling timing. The expected DeltaT update period is in milliseconds. The update period is normally the value that was used to configure the module's update time. If invalid, the instruction sets the appropriate bit in Status and disables RTSMissed checking. Valid = 1...32,767ms Default = 1
RTSTimeStamp	DINT	Module time stamp value for real time sampling timing. The time stamp value that corresponds to the

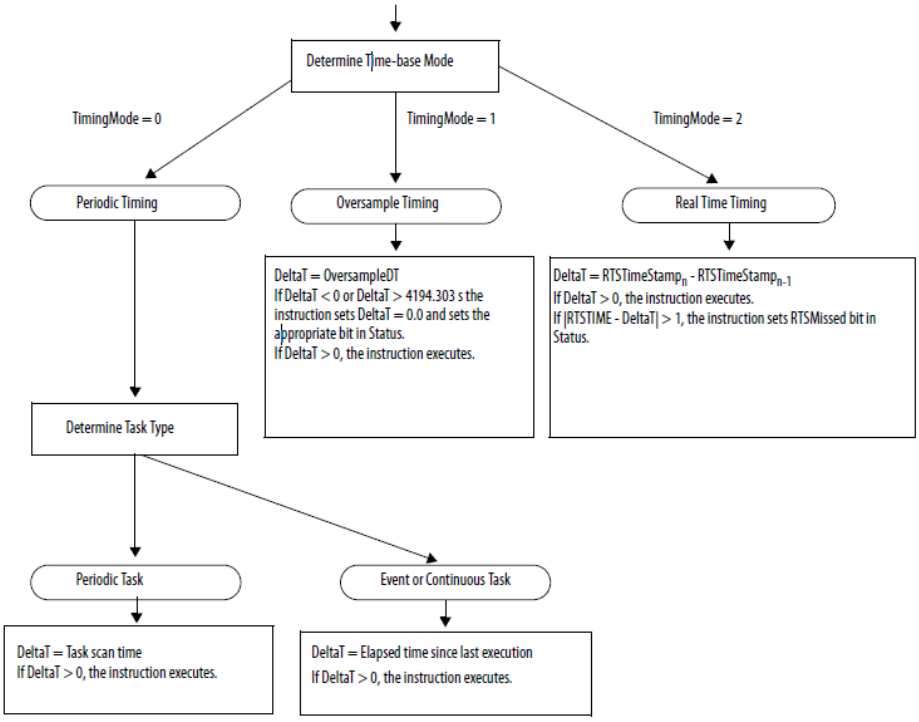
		last update of the input signal. This value is used to calculate DeltaT. If invalid, the instruction sets the appropriate bit in Status, disables execution of the control algorithm, and disables RTSMissed checking. Valid =0...32,767ms (wraps from 32767 to 0) 1 count = 1 millisecond Default = 0

Output Parameters

Output Parameter	Data Type	Description
DeltaT	REAL	Elapsed time between updates. This is the elapsed time in seconds used by the control algorithm to calculate the process output. Periodic: DeltaT = task scan rate if task is Periodic task, DeltaT = elapsed time since previous instruction execution if task is Event or Continuous task Oversample: DeltaT = OversampleDT Real Time Sampling: DeltaT = (RTSTimeStampn - RTSTimeStampn-1)
Status	DINT	Status of the function block.
TimingModeInv (Status.27)	BOOL	Invalid TimingMode value.
RTSMissed (Status.28)	BOOL	Only used in real time sampling mode. Set when ABS | DeltaT - RTSTime | > 1 (.001 second).
RTSTimeInv (Status.29)	BOOL	Invalid RTSTime value.
RTSTimeStampInv (Status.30)	BOOL	Invalid RTSTimeStamp value.
DeltaTInv (Status.31)	BOOL	Invalid DeltaT value.

Overview of Timing Modes

The following diagram shows how an instruction determines the appropriate timing mode.



Program/Operator Control

The following instructions support the concept of Program/Operator control.

- Enhanced Select (ESEL)
- Totalizer (TOT)
- Enhanced PID (PIDE)
- Ramp/Soak (RMPS)
- Discrete 2-State Device (D2SD)
- Discrete 3-State Device (D3SD)

Program/Operator control lets you control these instructions simultaneously from both your user program and from an operator interface device. When in Program control, the instruction is controlled by the Program inputs to the instruction; when in Operator control, the instruction is controlled by the Operator inputs to the instruction.

Program or Operator control is determined by using these inputs.

Input	Description
.ProgProgReq	A program request to go to Program control.
.ProgOperReq	A program request to go to Operator control.
.OperProgReq	An operator request to go to Program control.

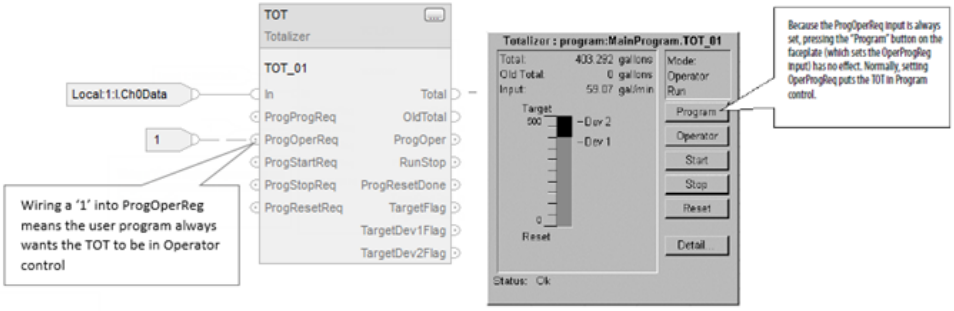
.OperOperReq	An operator request to go to Operator control.

To determine whether an instruction is in Program or Operator control, examine the ProgOper output. If ProgOper is set, the instruction is in Program control; if ProgOper is cleared, the instruction is in Operator control.

Operator control takes precedence over Program control if both input request bits are set. For example, if ProgProgReq and ProgOperReq are both set, the instruction goes to Operator control.

The Program request inputs take precedence over the Operator request inputs. This provides the capability to use the ProgProgReq and ProgOperReq inputs to 'lock' an instruction in a desired control.

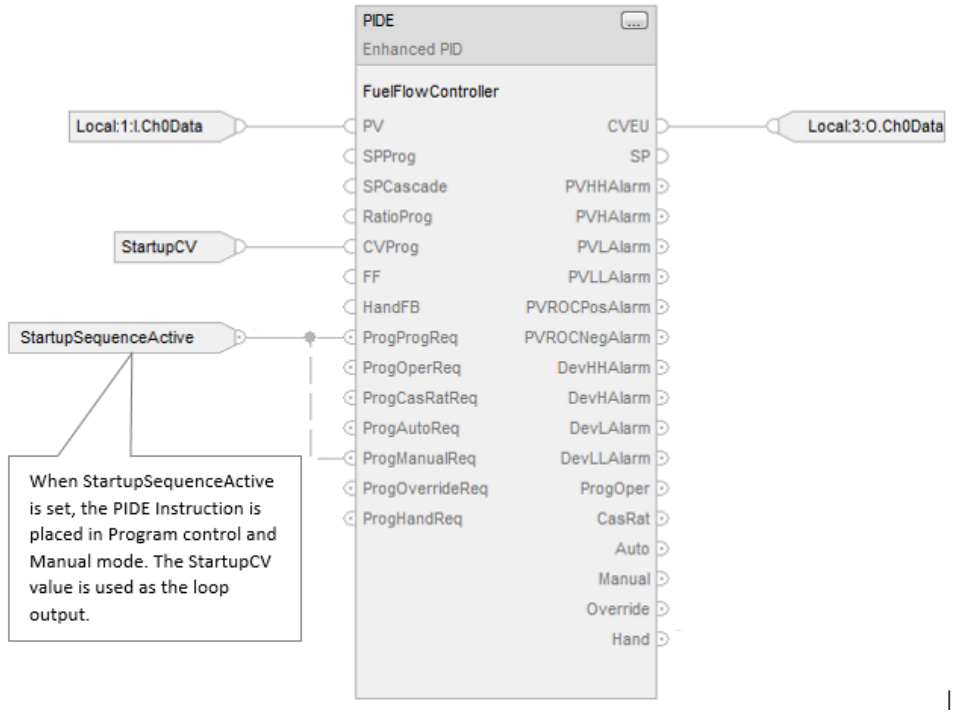
For example, let's assume that a Totalizer instruction will always be used in Operator control, and your user program will never control the running or stopping of the Totalizer. In this case, you could wire a literal value of 1 into the ProgOperReq. This would prevent the operator from ever putting the Totalizer into Program control by setting the OperProgReq from an operator interface device.



Likewise, constantly setting the ProgProgReq can 'lock' the instruction into Program control. This is useful for automatic startup sequences when you want the program to control the action of the instruction without worrying about an operator inadvertently taking control of the instruction.

In this example, you have the program set the ProgProgReq input during the startup, and then clear the ProgProgReq input once the startup was complete. Once the ProgProgReq input is cleared, the instruction remains in Program control until it receives a request to change. For example, the operator could set the OperOperReq input from a faceplate to take over control of that instruction.

The following example shows how to lock an instruction into Program control.

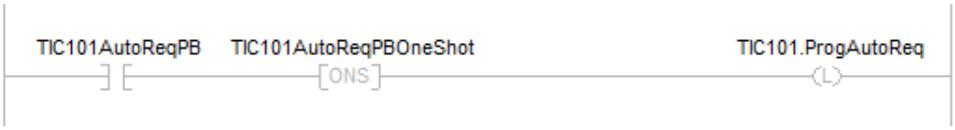


Operator request inputs to an instruction are always cleared by the instruction when it executes. This allows operator interfaces to work with these instructions by merely setting the desired mode request bit. You don't have to program the operator interface to reset the request bits. For example, if an operator interface sets the OperAutoReq input to a PIDE instruction, when the PIDE instruction executes, it determines what the appropriate response should be and clears the OperAutoReq.

Program request inputs are not normally cleared by the instruction because these are normally wired as inputs into the instruction. If the instruction clears these inputs, the input would just get set again by the wired input. There might be situations where you want to use other logic to set the Program requests in such a manner that you want the Program requests to be cleared by the instruction. In this case, you can set the ProgValueReset input and the instruction will always clear the Program mode request inputs when it executes.

In this example, a rung of ladder logic in another routine is used to one-shot latch a ProgAutoReq to a PIDE instruction when a push button is pushed.

When the TIC101AutoReq push button is pressed, one-shot latch ProgAutoReq for the PIDE instruction TIC101. TIC101 has been configured with the ProgValueReset input set. ProgAutoReq get reset because ProgValuieReset is always set.



Function Block States

Logix-based controllers evaluate function block instructions based on the state of different conditions:

Condition	Description
prescan	Prescan for function block routines is the same as for ladder diagram routines. The only difference is that the EnableIn parameter for each function block instruction is cleared during prescan.
instruction first scan	Instruction first scan refers to the first time an instruction is executed after prescan. The controller uses instruction first scan to read current inputs and determine the appropriate state to be in.
instruction first run	Instruction first run refers to the first time the instruction executes with a new instance of a data structure. The controller uses instruction first run to generate coefficients and other data stores that do not change for a function block after initial download.

Every function block instruction also included EnableIn and EnableOut parameters:

- Function block instructions execute normally when EnableIn is set.
- When EnableIn is cleared, the function block instruction either executes prescan logic, postscan logic, or simply skips normal algorithm execution.
- EnableOut mirrors EnableIn. However, if Function Block detects an overflow condition, EnableOut is also cleared.
- Function Block resumes from where it left off when EnableIn toggles from cleared to set. However, there are some function block instructions that specify special functionality (for example, re-initialization) when EnableIn toggles from cleared to set. For function block instructions with time base parameters, whenever the timing mode is Oversample, the instruction always resumes from where it left off when EnableIn toggles from cleared to set.

If the EnableIn parameter is not wired, the instruction always executes as normal and EnableIn remains set. If you clear EnableIn, it changes to set the next time the instruction executes.

Function Block Faceplate Controls

The Logix Designer programming application includes faceplate controls for some function block instructions. Faceplates are Active-X controls used in applications that acts as an Active-X container. The faceplates communicate with the controller via the RSLinx Classic OPC Server or the FactoryTalk Linx Gateway.

IMPORTANT: The Logix Designer programming application is not a valid Active-X container. An Active-X container is required to use the faceplates.

These instructions have faceplates:

- Process Discrete Input (PDI)
- Process Discrete Output (PDO)
- Process Analog Input (PAI)

- Process Analog Output (PAO)
- Alarm (ALM)
- Enhanced Select (ESEL)
- Totalizer (TOT)
- Ramp/Soak (RMPS)
- Discrete 2-State Device (D2SD)
- Discrete 3-State Device (D3SD)
- Enhanced PID (PIDE)

Configure the faceplates through property pages that open through container applications.

All faceplates have these property pages in common:

- General
- Display
- Font
- Locale

Faceplate Control - General

Use this tab to define/modify how the control operates.

Parameters

Communication

Select RSLinx Classic OPC Server or FactoryTalk Linx. If you select RSLinx Classic OPC Server, you must also specify:

- whether to launch remotely
- the access path to the remote machine

If you select FactoryTalk Linx, you must also specify the FactoryTalk Area.

Tag

Enter the name of a specific function block instruction to connect with this control.

Update Rate

Enter the Update Rate of the control in seconds. You can click the arrows to increment this value in 0.25 second increments. The default value is 1.00 second.

Faceplate Control - Display

Use this tab to define how the faceplate control will appear on your screen.

Parameters

Background Color

This button indicates the color of the faceplate's background. Select the button to change this color. The default color is light gray.

Show Frame

Select or clear this box, depending on whether you want to display a three-dimensional frame around this control. Use this option to separate the control from other items that might appear on your display. This option is selected by default.

OK

Select this button to accept edits and close the **Faceplate Control Properties** dialog.

Cancel

Select this button to cancel edits and close the **Faceplate Control Properties** dialog.

Apply

Select this button to apply edits and continue editing in the **Faceplate Control Properties** dialog.

Faceplate Control - Font

Use this tab to define the fonts that appear on the faceplates. From here, you can configure a ControlFont to be used in the main part of the faceplate, and a MinorFont to be used in scales and other minor portions of the faceplate.

Parameters

Property Name

Choose the font you want to configure from the pull-down menu. Choose from ControlFont or MinorFont; the default is ControlFont.

Font

Choose the font you wish to use for the control from the list of available fonts. The default font is Arial.

Style

Choose the style you wish to use for the control from the pull-down menu. The default style is Regular.

Size

Enter the point size you wish to use for this font. The default size of the ControlFont is 10.5 points; the default size of the MinorFont is 8.25 points.

Strikeout

Check this box if you want to use the strikeout effect, which draws a line through the font. This option is unchecked, by default.

Underline

Check this box if you want to use the underline effect, which draws a line below the font. This option is unchecked, by default.

OK

Click this button to accept your edits and close the Faceplate Control Properties dialog.

Cancel

Click this button to cancel your edits and close the Faceplate Control Properties dialog.

Apply

Click this button to apply your edits and continue editing in the Faceplate Control Properties dialog.

Faceplate Control - Locale

Use this tab to define the language requirements for the faceplates.

Parameters

Locale

Choose the language you want to use from the pull-down menu. Choose from:

- English
  - Portuguese
    - French
      - Italian
        - German
          - Spanish

OK

Click this button to accept your edits and close the Faceplate Control Properties dialog.

Cancel

Click this button to cancel your edits and close the Faceplate Control Properties dialog.

Apply

Click this button to apply your edits and continue editing in the Faceplate Control Properties dialog.

Structured Text Programming

These are the issues that are unique with structured text programming. Review the following topics to make sure you understand how your structured text programming executes.

Structured Text Syntax

Structured text is a textual programming language that uses statements to define what to execute.

- Structured text is not case sensitive.
- Use tabs and carriage returns (separate lines) to make your structured text easier to read. They have no effect on the execution of the structured text.

Structured text is not case sensitive. Structured text can contain these components.

Term	Definition	Examples
Assignment	Use an assignment statement to assign values to tags. The := operator is the assignment operator. Terminate the assignment with a semi colon ';.'	tag := expression;
Expression	An expression is part of a complete assignment or construct statement. An expression evaluates to a number (numerical expression), a String (string expression), or to a true or false state (BOOL expression)	
Tag Expression	A named area of the memory where data is stored (BOOL, SINT, INT, DINT, REAL, String).	value1

Immediate Expression	A constant value	4
Operators Expression	A symbol or mnemonic that specifies an operation within an expression.	tag1 + tag2 tag1 >= value1
Function Expression	When executed, a function yields one value. Use parentheses to contain the operand of a function. Even though their syntax is similar, functions differ from instructions in that functions can be used only in expressions. Instructions cannot be used in expressions.	function(tag1)

Instruction	An instruction is a standalone statement.
	An instruction uses parentheses to contain its operands.
	Depending on the instruction, there can be zero, one, or multiple operands.
	When executed, an instruction yields one or more values that are part of a data structure. Terminate the instruction with a semi colon(;).
	Even though their syntax is similar, instructions differ from functions in that instructions cannot be used in expressions. Functions can be used only in expressions.
		instruction();
		instruction(operand);
		instruction(operand1, operand2,operand3);

Construct	A conditional statement used to trigger structured text code (that is, other statements). Terminate the construct with a semi colon (;).	IF...THEN CASE FOR...DO WHILE...DO REPEAT...UNTIL EXIT
Comment	Text that explains or clarifies what a section of structured text does. Use comments to make it easier to interpret the structured text. Comments do not affect the execution of the structured text. Comments can appear anywhere in structured text.	//comment (*start of comment . . . end of comment*) /*start of comment . . . end of comment*/

Structured Text Components: Comments

To make your structured text easier to interpret, add comments to it.

- Comments let you use plain language to describe how your structured text works.
- Comments do not affect the execution of the structured text.

To add comments to your structured text:

To add a comment	Use one of these formats
on a single line	//comment (*comment*) /*comment*/
at the end of a line of structured text	
within a line of structured text	(*comment*) /*comment*/
that spans more than one line	(*start of comment. . .end of comment*) /*start of comment. . .end of comment*/

For example:

Format	Example

//comment	**At the beginning of a line**
	//Check conveyor belt direction
	IF conveyor_direction THEN...
	At the end of a line
	ELSE //If conveyor isn't moving, set alarm light
	light := 1;
	END_IF;

(*comment*)	Sugar.Inlet[:=]1;(*open the inlet*) IF Sugar.Low (*low level LS*)& Sugar.High (*high level LS*)THEN... (*Controls the speed of the recirculation pump. The speed depends on the temperature in the tank.*) IF tank.temp > 200 THEN...
/*comment*/	Sugar.Inlet:=0;/*close the inlet*/ IF bar_code=65 /*A*/ THEN... /*Gets the number of elements in the Inventory array and stores the value in the Inventory_Items tag*/ SIZE(Inventory,0,Inventory_Items);

Structured Text Components: Assignments

Use an assignment to change the value stored within a tag. An assignment has this syntax:

tag := expression;

where:

<table>
<tr><th>Component</th><th colspan="2">Description</th></tr>
<tr><td>Tag</td><td colspan="2">Represents the tag that is getting the new value; the tag must be a BOOL, SINT, INT, DINT, STRING, or REAL.

The STRING tag is only applicable to CompactLogix 5380, ControlLogix 5580, Compact GuardLogix 5380, GuardLogix 5580, and ControlLogix 5590 controllers.</td></tr>
<tr><td>:=</td><td colspan="2">Is the assignment symbol</td></tr>
<tr><td>Expression</td><td colspan="2">Represents the new value to assign to the tag</td></tr>
<tr><td></td><td>If tag is this data type</td><td>Use this type of expression</td></tr>
<tr><td></td><td>BOOL</td><td>BOOL</td></tr>
<tr><td></td><td>SINT
INT
DINT
REAL</td><td>Numeric</td></tr>
<tr><td></td><td>STRING
(CompactLogix 5380, ControlLogix 5580, Compact GuardLogix 5380, GuardLogix 5580, and ControlLogix 5590 controllers only.)</td><td>String type, including string tag and string literal
(CompactLogix 5380, ControlLogix 5580, Compact GuardLogix 5380, GuardLogix 5580, and ControlLogix 5590 controllers only.)</td></tr>
<tr><td>;</td><td colspan="2">Ends the assignment</td></tr>
</table>

The tag retains the assigned value until another assignment changes the value.

The expression can be simple, such as an immediate value or another tag name, or the expression can be complex and include several operators and functions, or both. Refer to Expressions for more information.

I/O module data updates asynchronously to the execution of logic. If you reference an input multiple times in your logic, the input could change state between separate references. If you need the input to have the same state for each reference, buffer the input value and reference that buffer tag. For more information, see Logix 5000 Controllers Common Procedures, publication 1756-PM001.You can also use Input and Output program parameters which automatically buffer the data during the Logix Designer application execution. See LOGIX 5000 Controllers Program Parameters Programming Manual, publication 1756-PM021.

Assign an ASCII character to a string data member

Use the assignment operator to assign an ASCII character to an element of the DATA member of a string tag. To assign a character, specify the value of the character or specify the tag name, DATA member, and element of the character. For example:

This is OK	This is not OK
string1.DATA[0] := 65;	string1.DATA[0] := A;
string1.DATA[0]:= string2.DATA[0];	string1 := string2; This assigns all content of string2 to string1 instead of just one character.

To add or insert a string of characters to a string tag, use either of these ASCII string instructions:

To	Use this instruction
Add characters to the end of a string	CONCAT
Insert characters into a string	INSERT

Specify a non-retentive assignment

The non-retentive assignment is different from the regular assignment described above in that the tag in a non-retentive assignment is reset to zero each time the controller:

- Enters the Run mode
- Leaves the step of an SFC if you configure the SFC for Automatic reset. This applies only if you embed the assignment in the action of the step or use the action to call a structured text routine by using a JSR instruction.

A non-retentive assignment has this syntax:

tag [:=] expression ;

where:

Component	Description
tag	Represents the tag that is getting the new value; the tag must be a BOOL, SINT, INT, DINT, STRING, or REAL. The STRING tag is only applicable to CompactLogix 5380, ControlLogix 5580, Compact GuardLogix 5380, GuardLogix 5580, and ControlLogix 5590 controllers.
[:=]	Is the non-retentive assignment symbol.
expression	Represents the new value to assign to the tag.

	If tag is this data type	Use this type of expression
	BOOL	BOOL
	SINT	Numeric
	INT	
	DINT	
	REAL	
	STRING (CompactLogix 5380, ControlLogix 5580, Compact GuardLogix 5380, GuardLogix 5580, and ControlLogix 5590 controllers only.)	String type, including string tag and string literal (CompactLogix 5380, ControlLogix 5580, Compact GuardLogix 5380, GuardLogix 5580, and ControlLogix 5590 controllers only.)

Structured Text Components: Expressions

An expression is a tag name, equation, or comparison. To write an expression, use any of the following:

- Tag name that stores the value (variable)
- Number that you enter directly into the expression (immediate value)
- String literal that you enter directly into the expression (CompactLogix 5380, ControlLogix 5580, Compact GuardLogix 5380, and GuardLogix 5580 controllers

  only)
- Functions, such as: ABS, TRUNC
- Operators, such as: +, -, <, >, And, Or

Follow these guidelines for writing expressions:

- Use any combination of upper-case and lower-case letter. For example, these variations of "AND" are acceptable: AND, And, and.
- For more complex requirements, use parentheses to group expressions within expressions. This makes the whole expression easier to read, and ensures that the expression executes in the desired sequence.

Use these expressions for structured text:

BOOL expression: An expression that produces the BOOL value of 1 (true) or 0 (false).

- A bool expression uses bool tags, relational operators, and logical operators to compare values or check if conditions are true or false. For example, tag1>65.
- A simple bool expression can be a single BOOL tag.
- Typically, use bool expressions to condition the execution of other logic.

Numeric expression: An expression that calculates an integer or floating-point value.

- A numeric expression uses arithmetic operators, arithmetic functions, and bitwise operators. For example, tag1+5.
- Nest a numeric expression within a BOOL expression. For example, (tag1+5)>65.

String expression: An expression that represents a string

- A simple expression can be a string literal or a string tag

Use this table to select the operators for expressions.

If	Use
Calculating an arithmetic value	Arithmetic operators and functions
Comparing two values or strings	Relational operators
Verifying if conditions are true or false	Logical operators
Comparing the bits within values	Bitwise operators

Use arithmetic operators and functions

Combine multiple operators and functions in arithmetic expressions.

Operators calculate new values.

To	Use this operator	Optimal data type
Add	+	DINT, REAL
Subtract/negate	-	DINT, REAL
Multiply	*	DINT, REAL
Exponent (x to the power of y)	**	DINT, REAL
Divide	/	DINT, REAL
Modulo-divide	MOD	DINT, REAL

Functions perform math operations. Specify a constant, a non-Boolean tag, or an expression for the function.

For	Use this function	Optimal data type
Absolute value	ABS (numeric_expression)	DINT, REAL
Arc cosine	ACOS (numeric_expression)	REAL
Arc sine	ASIN (numeric_expression)	REAL
Arc tangent	ATAN (numeric_expression)	REAL

Cosine	COS (numeric_expression)	REAL
Radians to degrees	DEG (numeric_expression)	DINT, REAL
Natural log	LN (numeric_expression)	REAL
Log base 10	LOG (numeric_expression)	REAL
Degrees to radians	RAD (numeric_expression)	DINT, REAL
Sine	SIN (numeric_expression)	REAL
Square root	SQRT (numeric_expression)	DINT, REAL
Tangent	TAN (numeric_expression)	REAL
Truncate	TRUNC (numeric_expression)	DINT, REAL

The table provides examples for using arithmetic operators and functions.

Use this format	Example	
	For this situation	Write
value1 operator value2	If gain_4 and gain_4_adj are DINT tags and your specification says: 'Add 15 to gain_4 and store the result in gain_4_adj'"	gain_4_adj := gain_4+15;
operator value1	If alarm and high_alarm are DINT tags and your specification says: 'Negate high_alarm and store the result in alarm.'	alarm:= -high_alarm;
function(numeric_expression)	If overtravel and overtravel_POS are DINT tags and your specification says: 'Calculate the absolute value of overtravel and store the result in overtravel_POS.'	overtravel_POS := ABS(overtravel);
value1 operator (function((value2+value3)/2)	If adjustment and position are DINT tags and sensor1 and sensor2 are REAL tags and your specification says: 'Find the absolute value of the average of sensor1 and sensor2, add the adjustment, and store the result in position.'	position := adjustment + ABS((sensor1 + sensor2)/2);

Use bitwise operators

Bitwise operators manipulate the bits within a value based on two values.

The following provides an overview of the bitwise operators.

For	Use this operator	Optimal data type
bitwise AND	&, AND	DINT
bitwise OR	OR	DINT
bitwise exclusive OR	XOR	DINT
bitwise complement	NOT	DINT

This is an example.

Use this format	Example	
	For this situation	Use
value1 operator value2	If input1, input2, and result1 are DINT tags and your specification says: "Calculate the bitwise result of input1 and input2. Store the result in result1."	result1 := input1 AND input2;

Use logical operators

Use logical operators to verify if multiple conditions are true or false. The result of a logical operation is a BOOL value.

If the comparison is	The result is
true	1
false	0

Use these logical operators.

For this comparison	Use this operator	Optimal data type
logical AND	&, AND	BOOL
logical OR	OR	BOOL
logical exclusive OR	XOR	BOOL
logical complement	NOT	BOOL

The table provides examples of using logical operators.

Use this format	Example	
	For this situation	Use

BOOLtag	If photoeye is a BOOL tag and your specification says: "If photoeye_1 is on then..."	IF photoeye THEN...
NOT BOOLtag	If photoeye is a BOOL tag and your specification says: "If photoeye is off then..."	IF NOT photoeye THEN...
expression1 & expression2	If photoeye is a BOOL tag, temp is a DINT tag, and your specification says: "If photoeye is on and temp is less than 100 then..."	IF photoeye & (temp<100) THEN...
expression1 OR expression2	If photoeye is a BOOL tag, temp is a DINT tag, and your specification says: "If photoeye is on or temp is less than 100 then...".	IF photoeye OR (temp<100) THEN...
expression1 XOR expression2	If photoeye1 and photoeye2 are BOOL tags and your specification says: "If: photoeye1 is on while photoeye2 is off or photoeye1 is off while photoeye2 is on then..."	IF photoeye1 XOR photoeye2 THEN...
BOOLtag := expression1 & expression2	If photoeye1 and photoeye2 are BOOL tags, open is a BOOL tag, and your specification says: "If photoeye1 and photoeye2 are both on, set open to true"	open := photoeye1 & photoeye2;

Use relational operators

Relational operators compare two values or strings to provide a true or false result. The result of a relational operation is a BOOL value.

If the comparison is	The result is
True	1
False	0

Use these relational operators.

For this comparison	Use this operator	Optimal data type
Equal	=	DINT, REAL, String type
Less than	<	DINT, REAL, String type

Less than or equal	<=	DINT, REAL, String type
Greater than	>	DINT, REAL, String type
Greater than or equal	>=	DINT, REAL, String type
Not equal	<>	DINT, REAL, String type

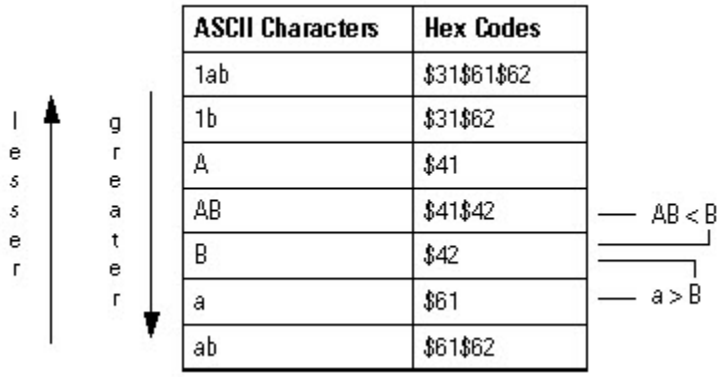
The table provides examples of using relational operators

Use this format	Example	
	For this situation	Write
value1 operator value2	If temp is a DINT tag and your specification says: 'If temp is less than 100· then…'	IF temp<100 THEN…
stringtag1 operator stringtag2	If bar_code and dest are string tags and your specification says: 'If bar_code equals dest then…'	IF bar_code=dest THEN…
stringtag1 operator 'character string literal'	If bar_code is a string tag and your specification says: 'If bar_code equals 'Test PASSED' then…'	IF bar_code='Test PASSED' THEN…
char1 operator char2 To enter an ASCII character directly into the expression, enter the decimal value of the character.	If bar_code is a string tag and your specification says: 'If bar_code.DATA[0] equals 'A' then…'	IF bar_code.DATA[0]=65 THEN…
bool_tag := bool_expressions	If count and length are DINT tags, done is a BOOL tag, and your specification says: 'If count is greater than or equal to length, you are done counting.'	Done := (count >= length);

How strings are evaluated

The hexadecimal values of the ASCII characters determine if one string is less than or greater than another string.

- When the two strings are sorted as in a telephone directory, the order of the strings determines which one is greater.



ASCII Characters	Hex Codes
1ab	$31$61$62
1b	$31$62
A	$41
AB	$41$42
B	$42
a	$61
ab	$61$62

- Strings are equal if their characters match.
- Characters are case sensitive. Upper case "A" ($41) is not equal to lower case "a" ($61).

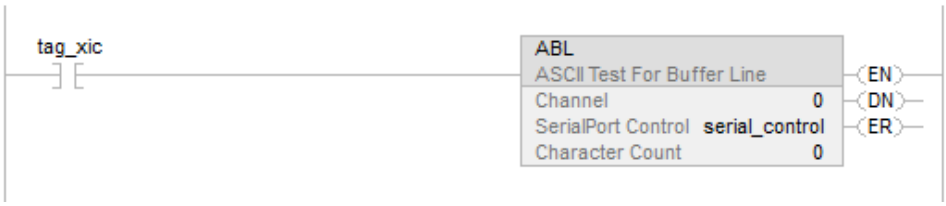
Structured Text Components: Instructions

Structured text statements can also be instructions. A structured text instruction executes each time it is scanned. A structured text instruction within a construct executes every time the conditions of the construct are true. If the conditions of the construct are false, the statements within the construct are not scanned. There is no rung-condition or state transition that triggers execution.

This differs from function block instructions that use EnableIn to trigger execution. Structured text instructions execute as if EnableIn is always set.

This also differs from ladder diagram instructions that use rung-condition-in to trigger execution. Some ladder diagram instructions only execute when rung- condition-in toggles from false to true. These are transitional ladder diagram instructions. In structured text, instructions execute when they are scanned unless pre-conditioning the execution of the structured text instruction.

For example, the ABL instruction is a transitional instruction in ladder diagram. In this example, the ABL instruction only executes on a scan when tag_xic transitions from cleared to set. The ABL instruction does not execute when tag_xic stays set or when tag_xic clears.



In structured text, if writting this example as:

IF tag_xic THEN ABL(0,serial_control);

END_IF;

The ABL instruction will execute every scan that tag_xic is set, not just when tag_xic transitions from cleared to set.

If you want the ABL instruction to execute only when tag_xic transitions from cleared to set, you have to condition the structured text instruction. Use a one-shot to trigger execution.

osri_1.InputBit := tag_xic;

```
OSRI(osri_1);
IF (osri_1.OutputBit) THEN
ABL(0,serial_control);
END_IF;
```

Structured Text Components: Constructs

Program constructs alone or nest within other constructs.

If	Use this construct
Doing something if or when specific conditions occur	IF. . . THEN
Selecting what to do based on a numerical value	CASE. . . OF
Doing something a specific number of times before doing anything else	FOR. . . DO
Continuing doing something when certain conditions are true	WHILE. . . DO
Continuing doing something until a condition is true	REPEAT. . . UNTIL

Some Key Words are Reserved

These constructs are not available:

- GOTO
- REPEAT

Logix Designer application will not let you use them as tag names or constructs.

Character string literals

Character string literals include single byte or double byte encoded characters. A single-byte string literal is a sequence of zero or more characters that are prefixed and terminated by the single quote character ('). In single byte character strings, the three-character combination of the dollar sign ($) followed by two hexadecimal digits is interpreted as the hexadecimal representation of the eight-bit character code as shown in the following table.

Character string literals are only applicable to the CompactLogix 5380, ControlLogix 5580, Compact GuardLogix 5380, GuardLogix 5580, and ControlLogix 5590 controllers. Studio 5000 only supports single byte characters.

Character string literals

No.	Description	Example
1a	Empty string (length zero)	''

1b	String of length one or character CHAR containing a single character	'A'
1c	String of length one or character CHAR containing the “space” character	' '
1d	String of length one or character CHAR containing the “single quote” character	'$''
1e	String of length one or character CHAR containing the “double quote” character	'"'
1f	Support of two character combinations	'RL'
1g	Support of a character representation with ‘$’ and two hexadecimal characters	'$0A'

Two-character combinations in character strings

No.	Description	Example
1	Dollar sign	$$
2	Single quote	$'
3	Line feed	$L or $l
4	Newline	$N or $n
5	Form feed (page)	$P or $p
6	Carriage return	$R or $r
7	Tabulator	$T or $t

The newline character provides an implementation-independent means of defining the end of a line of data for both physical and file I/O; for printing, the effect is that of ending a line of data and resuming printing at the beginning of the next line.The $' combination is only valid inside single quoted string literals.

Integer literal suffixes

This table lists suffixes you can add to integer literals in Structured Text, and the corresponding range for each suffix.

Suffix	Literal data type	Range
None	DINT	-2,147,483,648 to 2,147,483,648
L	LINT	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,808
U	UDINT	0 to 4,294,967,295
UL	ULINT	0 to 18,446,744,073,709,551,615

CASE_OF

Use CASE_OF to select what to do based on a numerical value.

Operands

CASE numeric_expression OF

selector1: statement;

selectorN: statement; ELSE

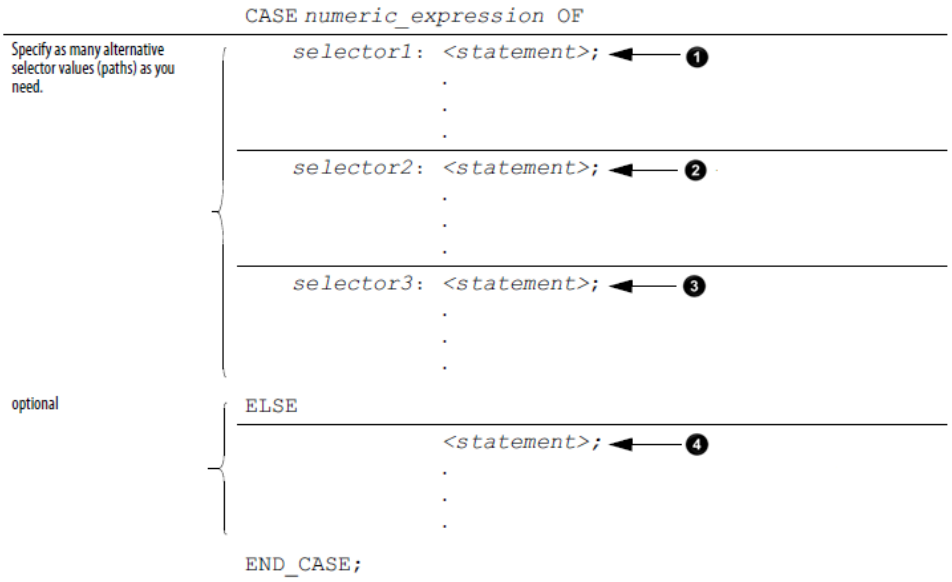
Structured Text

Operand	Type	Format	Enter
Numeric_ expression	SINT INT DINT REAL	Tag expression	Tag or expression that evaluates to a number (numeric expression)
Selector	SINT INT DINT REAL	Immediate	Same type as numeric_expression

IMPORTANT: If using REAL values, use a range of values for a selector because a REAL value is more likely to be within a range of values than an exact match of one, specific value.

Description

The syntax is described in the table.



These are the syntax for entering the selector values.

When selector is	Enter
One value	value: statement
Multiple, distinct values	value1, value2, valueN : <statement> Use a comma (,) to separate each value.
A range of values	value1..valueN : <statement> Use two periods (..) to identify the range.
Distinct values plus a range of values	valuea, valueb, value1..valueN : <statement>

The CASE construct is similar to a switch statement in the C or C++ programming languages. With the CASE construct, the controller executes only the statements that associated with the first matching selector value. Execution always breaks after the statements of that selector and goes to the END_CASE statement.

Affects Math Status Flags

No

Major/Minor Faults

None

Example

If you want this	Enter this structured text

If recipe number = 1 then Ingredient A outlet 1 = open (1) Ingredient B outlet 4 = open (1)	CASE recipe_number OF
	1: Ingredient_A.Outlet_1 :=1; Ingredient_B.Outlet_4 :=1;
If recipe number = 2 or 3 then Ingredient A outlet 4 = open (1) Ingredient B outlet 2 = open (1)	2,3: Ingredient_A.Outlet_4 :=1; Ingredient_B.Outlet_2 :=1;
If recipe number = 4, 5, 6, or 7 then Ingredient A outlet 4 = open (1) Ingredient B outlet 2 = open (1)	4..7: Ingredient_A.Outlet_4 :=1; Ingredient_B.Outlet_2 :=1;
If recipe number = 8, 11, 12, or 13 then Ingredient A outlet 1 = open (1) Ingredient B outlet 4 = open (1)	8,11..13 Ingredient_A.Outlet_1 :=1; Ingredient_B.Outlet_4 :=1;
Otherwise all outlets = closed (0)	ELSE
	Ingredient_A.Outlet_1 [:=]0; Ingredient_A.Outlet_4 [:=]0; Ingredient_B.Outlet_2 [:=]0; Ingredient_B.Outlet_4 [:=]0;
	END_CASE;

The [:=] tells the controller to also clear the outlet tags whenever the controller does the following:

Enters the RUN mode.

Leaves the step of an SFC if configuring the SFC for Automatic reset. This applies only embedding the assignment in the action of the step or using the action to call a structured text routine via a JSR instruction.

FOR_DO

Use the FOR_DO loop to perform an action a number of times before doing anything else.

When enabled, the FOR instruction repeatedly executes the Routine until the Index value exceeds the Terminal value. The step value can be positive or negative. If it is negative, the loop ends when the index is less than the terminal value.. If it is positive, the loop ends when the index is greater than the terminal value.

Each time the FOR instruction executes the routine, it adds the Step size to the Index.

Do not loop too many times in a single scan. An excessive number of repetitions causes the controller watchdog to timeout and causes a major fault.

Operands

FOR count:= initial_value TO

final_value BY increment DO

<statement>;

END_FOR;

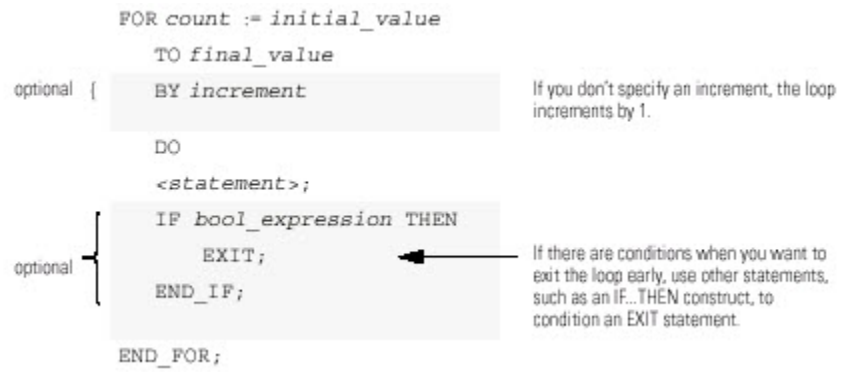
Operand	Type	Format	Description
count	SINT INT DINT	Tag	Tag to store count position as the FOR_DO executes
initial_ value	SINT INT DINT	Tag expression Immediate	Must evaluate to a number Specifies initial value for count
final_ value	SINT INT DINT	Tag expression Immediate	Specifies final value for count, which determines when to exit the loop
increment	SINT INT DINT	Tag expression Immediate	(Optional) amount to increment count each time through the loop If you don't specify an increment, the count increments by 1.

IMPORTANT:

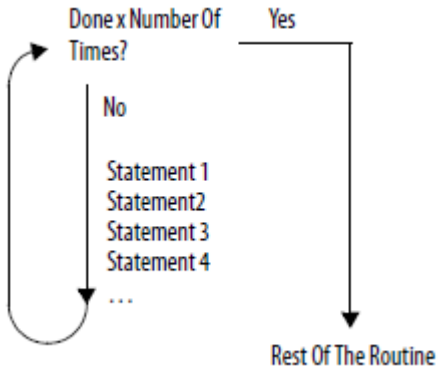
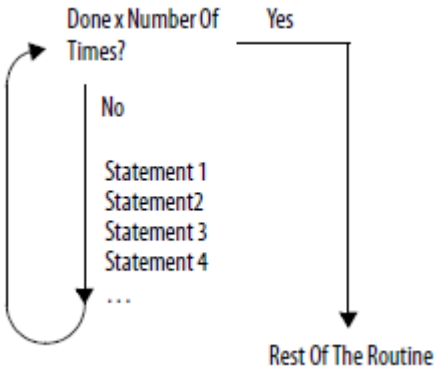
- Do not iterate within the loop too many times in a single scan.
- The controller does not execute other statements in the routine until it completes the loop.
- A major fault occurs when completing the loop takes longer than the watchdog timer for the task.
- Consider using a different construct, such as IF_THEN.

Description

The syntax is described in the table.



This diagrams illustrates how a FOR_DO loop executes, and how an EXIT statement leaves the loop early.

	
The FOR_DO loop executes a specific number of times.	To stop the loop before the count reaches the last value, use an EXIT statement.

Affects Math Status Flags

No

Major/Minor Faults

A major fault will occur if	Fault type	Fault code
The construct loops too long.	6	1

Example 1

If performing the following,	Enter this structured text

Clear bits 0...31 in an array of BOOLs:
Initialize the subscript tag to 0.
Clear i . For example, when subscript = 5, clear array[5].
Add 1 to subscript.
If subscript is ≤ to 31, repeat 2 and 3.
Otherwise, stop.
	For subscript:=0 to 31 by 1 do
	array[subscript] := 0;
	End_for;

Example 2

If performing the following,	Enter this structured text

A user-defined data type (structure) stores the following information about an item in your inventory:
• Barcode ID of the item (String data type)
• Quantity in stock of the item (DINT data type)
An array of the above structure contains an element for each different item in your inventory. You want to search
	SIZE(Inventory,0,Inventory_Items);
	For position:=0 to Inventory_Items - 1 do
	If Barcode = Inventory[position].ID then
	Quantity := Inventory[position].Qty;

the array for a specific product (use its bar code) and determine the quantity that is in stock. 1. Get the size (number of items) of the Inventory array and store the result in 2. Inventory_Items (DINT tag). Initialize the position tag to 0. 1. If Barcode matches the ID of an item in the array, then: Set the Quantity tag = Inventory[position].Qty. This produces the quantity in stock of the item. Stop. Barcode is a string tag that stores the bar code of the item for which you are searching. For example, when position = 5, compare Barcode to Inventory[5].ID. 1. Add 1 to position. 2. If position is ≤ to (Inventory_Items -1), repeat 3 and 4. Since element numbers start at 0, the last element is 1 less than the number of elements in the array. Otherwise, stop.	Exit; End_if; End_for;

IF_THEN

Use IF_THEN to complete an action when specific conditions occur.

Operands

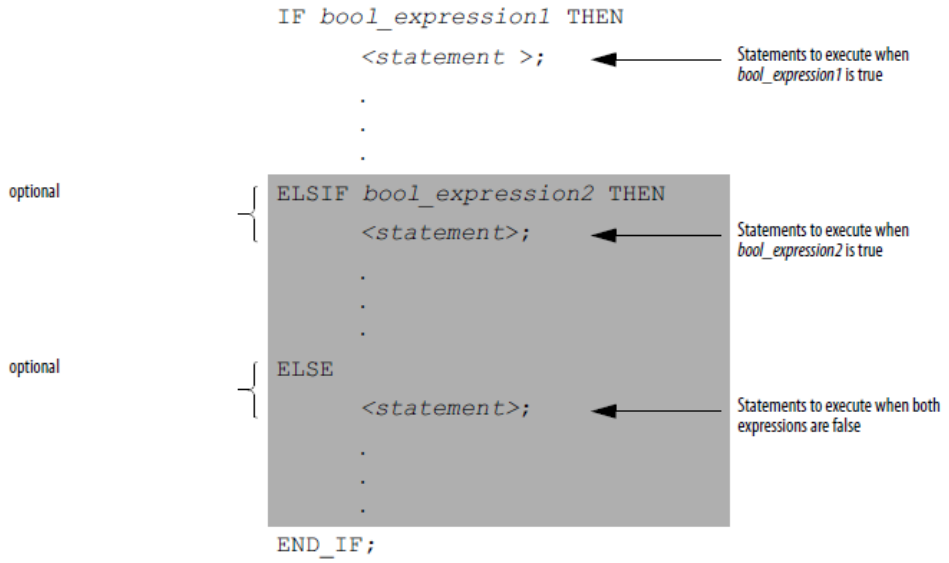
IF bool_expression THEN

<statement>;

Operand	Type	Format	Enter
Bool_ expression	BOOL	Tag expression	BOOL tag or expression that evaluates to a BOOL value (BOOL expression)

Description

The syntax is described in the table.



To use ELSIF or ELSE, follow these guidelines.

To select from several possible groups of statements, add one or more ELSIF statements.

Each ELSIF represents an alternative path.

Specify as many ELSIF paths as you need.

The controller executes the first true IF or ELSIF and skips the rest of the ELSIFs and the ELSE.

To do something when all of the IF or ELSIF conditions are false, add an ELSE statement.

The table summarizes different combinations of IF, THEN, ELSIF, and ELSE.

If	And	Use this construct
Doing something if or when conditions are true	Do nothing if conditions are false	IF_THEN
	Do something else if conditions are false	IF_THEN_ELSE
Selecting alternative statements or groups of statements based on input conditions	Do nothing if conditions are false	IF_THEN_ELSIF
	Assign default statements if all conditions are false	IF_THEN_ELSIF_ELSE

Affects Math Status Flags

No

Major/Minor Faults

None.

Examples

Example 1

IF...THEN

If performing this	Enter this structured text
IF rejects > 3 then	IF rejects > 3 THEN
conveyor = off (0)	conveyor := 0;
alarm = on (1)	alarm := 1;
	END_IF;

Example 2

IF_THEN_ELSE

If performiing this	Enter this structured text
If conveyor direction contact = forward (1) then	IF conveyor_direction THEN
light = off	light := 0;
Otherwise light = on	ELSE
	light [:=] 1;
	END_IF;

The [:=] tells the controller to clear light whenever the controller does the following :

Enters the RUN mode.

Leaves the step of an SFC if you configure the SFC for Automatic reset. (This applies only if you embed the assignment in the action of the step or use the action to call a structured text routine via a JSR instruction.)

Example 3

IF...THEN...ELSIF

If performing this	Enter this structured text
If sugar low limit switch = low (on) and sugar high limit switch = not high (on) then	IF Sugar.Low & Sugar.High THEN
inlet valve = open (on)	Sugar.Inlet [:=] 1;
Until sugar high limit switch = high (off)	ELSIF NOT(Sugar.High) THEN

	Sugar.Inlet := 0;
	END_IF;

The [:=] tells the controller to clear Sugar.Inlet whenever the controller does the following :

Enters the RUN mode.

Leaves the step of an SFC if you configure the SFC for Automatic reset. (This applies only if you embed the assignment in the action of the step or use the action to call a structured text routine via a JSR instruction.)

Example 4

IF...THEN...ELSIF...ELSE

If performing this	Enter this structured text
If tank temperature > 100	IF tank.temp > 200 THEN
then pump = slow	pump.fast :=1; pump.slow :=0; pump.off :=0;
If tank temperature > 200	ELSIF tank.temp > 100 THEN
then pump = fast	pump.fast :=0; pump.slow :=1; pump.off :=0;
Otherwise pump = off	ELSE
	pump.fast :=0; pump.slow :=0; pump.off :=1;
	END_IF;

REPEAT_UNTIL

Use the REPEAT_UNTIL loop to continue performing an action until conditions are true.

Operands

REPEAT

<statement>;

Structured Text

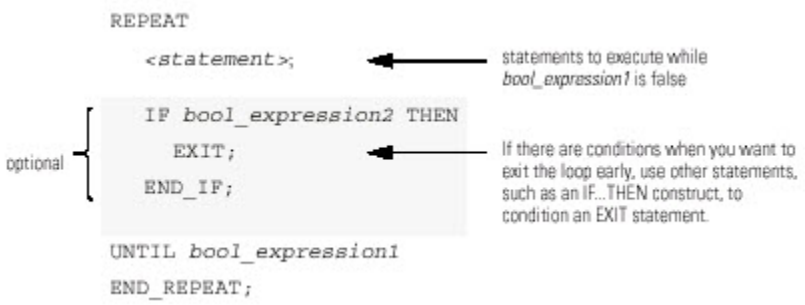
Operand	Type	Format	Enter
bool_ expression	BOOL	Tag expression	BOOL tag or expression that evaluates to a BOOL value (BOOL expression)

IMPORTANT:

- Do not iterate within the loop too many times in a single scan.
- The controller does not execute other statements in the routine until it completes the loop.
- A major fault occurs when completing the loop takes longer than the watchdog timer for the task.
- Consider using a different construct, such as IF_THEN.

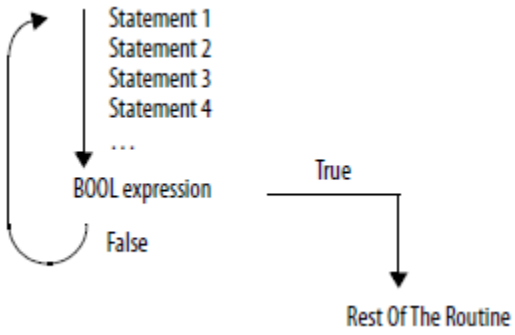
Description

The syntax is:

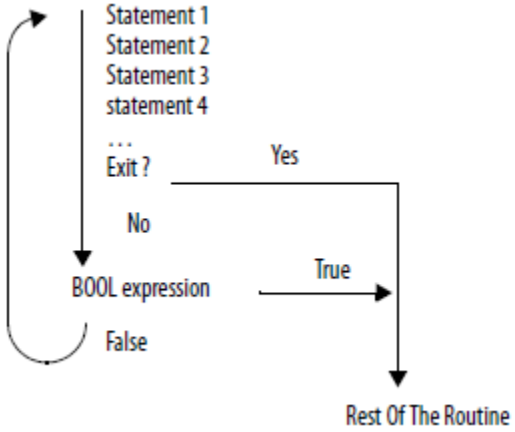


The following diagrams show how a REPEAT_UNTIL loop executes and how an EXIT statement leaves the loop early.

While the bool_expression is false, the controller executes only the statements within the REPEAT_UNTIL loop.



To stop the loop before the conditions are false, use an EXIT statement.



Affects Math Status Flags

No

Fault Conditions

A major fault will occur if	Fault type	Fault code
The construct loops too long	6	1

Example 1

If performing the following,	Enter this structured text

The REPEAT_UNTIL loop executes the statements in the construct and then determines if the conditions are true before executing the statements again. This differs from the WHILE_DO loop because the WHILE_DO The WHILE_DO loop evaluates its conditions first.
If the conditions are true, the controller then executes the statements within the loop. The statements in a REPEAT_UNTIL loop are always executed at least once. The statements in a WHILE_DO loop might never be executed.
	pos := -1;
	REPEAT
	pos := pos + 2;
	UNTIL ((pos = 101) OR (structarray[pos].value = targetvalue))
	end_repeat;

Example 2

If performing the following,	Enter this structured text

Move ASCII characters from a SINT array into a string tag. (In a SINT array, each element holds one character.) Stop when you reach the carriage return.
Initialize Element_number to 0.
Count the number of elements in SINT_array (array that contains the ASCII characters) and store the result in SINT_array_size (DINT tag).
Set String_tag[element_number] = the character at SINT_array[element_number].
Add 1 to element_number. This lets the controller check the next character in SINT_array.
Set the Length member of String_tag = element_number. (This records the number of characters in String_tag so far.)
If element_number = SINT_array_size, then stop. (You are at the end of the array and it does not contain a carriage return.)
	element_number := 0;
	SIZE(SINT_array, 0, SINT_array_size);
	Repeat
	String_tag.DATA[element_number] := SINT_array[element_number];
	element_number := element_number + 1;
	String_tag.LEN := element_number;
	If element_number = SINT_array_size then
	exit;
	end_if;
	Until SINT_array[element_number] = 13
	end_repeat;

If the character at SINT_array[element_number] = 13 (decimal value of the carriage return), then stop.	

WHILE_DO

Use the WHILE_DO loop to continue performing an action while certain conditions are true.

Operands

WHILE bool_expression DO

<statement>;

Structured Text

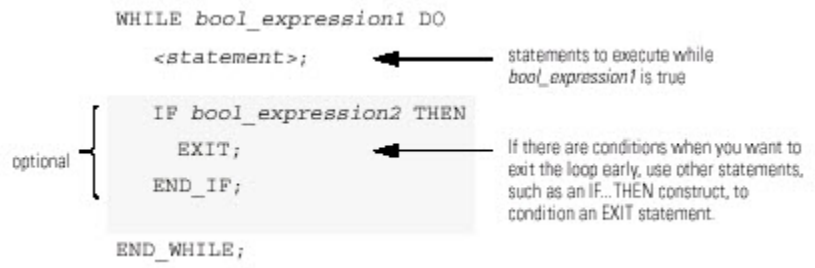
Operand	Type	Format	Description
bool_expression	BOOL	tag expression	BOOL tag or expression that evaluates to a BOOL value

IMPORTANT:

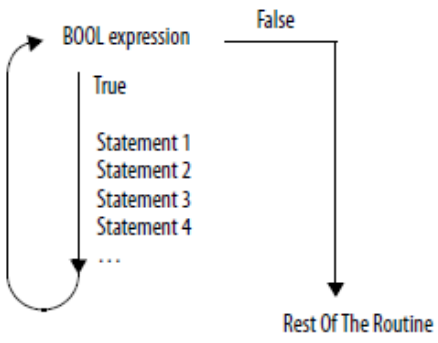
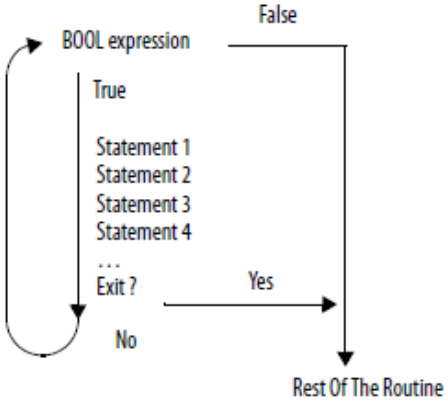
- Do not iterate within the loop too many times in a single scan.
- The controller does not execute any other statements in the routine until it completes the loop.
- A major fault occurs when completing the loop takes longer than the watchdog timer for the task.
- Consider using a different construct, such as IF_THEN.

Description

The syntax is:



The following diagrams illustrate how a WHILE_DO loop executes, and how an EXIT statement leaves the loop early.

	
While the bool_expression is true, the controller executes only the statements within the WHILE_DO loop.	To stop the loop before the conditions are true, use an EXIT statement.

Affects Math Status Flags

No

Fault Conditions

A major fault will occur if	Fault type	Fault code
the construct loops too long	6	1

Example 1

If performing the following,	Enter this structured text	
The WHILE_DO loop evaluates its conditions first. If the conditions are true, the controller then executes the statements within the loop. This differs from the REPEAT_UNTIL loop because the REPEAT_UNTIL loop executes the statements in the construct and then determines if the conditions are true before executing the statements again. The statements in a REPEAT_UNTIL loop are always executed at least once. The statements in a WHILE_DO loop might never be executed.	pos := 0;	
	While ((pos <= 100) & structarray[pos].value <> targetvalue)) do	
		pos := pos + 2;
		String_tag.DATA[pos] := SINT_array[pos];
	end_while;	

Example 2

If performing the following,	Enter this structured text

Move ASCII characters from a SINT array into a string tag. (In a SINT array, each element holds one character.) Stop when you reach the carriage return.
 	
Initialize Element_number to 0.
 	
Count the number of elements in SINT_array (array that contains the ASCII characters) and store the result in SINT_array_size (DINT tag).
 	
If the character at SINT_array[element_number] = 13 (decimal value of the carriage return), then stop.
 	
Set String_tag[element_number] = the character at SINT_array[element_number].
 	
Add 1 to element_number. This lets the controller check the next character in SINT_array.
 	
Set the Length member of String_tag = element_number. (This records the number of characters in String_tag so far.)
 	
If element_number = SINT_array_size, then stop. (You are at the end of the array and it does not contain a carriage return.)
	element_number := 0;
	SIZE(SINT_array, 0, SINT_array_size);
	While SINT_array[element_number] <> 13 do
	String_tag.DATA[element_number] := SINT_array[element_number];
	element_number := element_number + 1;
	String_tag.LEN := element_number;
	If element_number = SINT_array_size then
	exit;
	end_if;
	end_while;

Common Attributes for Advanced Process Control and Drives Instructions

Follow the guidelines in this chapter for the common attributes for the Advanced Process Control and Drives Instructions.

Common Attributes

For more information on attributes that are common to the Logix 5000™ instructions, see:

- Math status flags on page 591
- Immediate values on page 593
- Index through arrays on page 600
- Data conversions on page 593
- Elementary data types on page 596
- Floating Point Values on page 599
- Bit Addressing on page 601

Math status flags

Follow these guidelines for Math Status Flags.

Description

A set of Math Status Flags for accessing directly with instructions. These flags are only updated in ladder diagram routines, and are not tags, and flag aliases are not applicable.

Status Flags

This table describes that specific status flags.

Status Flag	Description
S:FS First scan flag	The first scan flag is set by the controller: • The first time a program is scanned after the controller goes to Run mode • The first time a program is scanned after the program is uninhibited • When a routine is called from an SFC Action and the step that owns that Action is first scanned. Use the first scan flag to initialize data for use in later scans. It is also referred to as the first pass bit.
S:N Negative flag	The controller sets the negative flag when the result of a math or logical operation is a negative value. Use this flag as a quick test for a negative value.

Status Flag	Description
S:Z Zero flag	The zero flag is set by the controller when the result of a math or logical operation is zero. Use this flag as a quick test for a zero value. The zero flag clears at the start of executing an instruction capable of setting this flag.
S:V Overflow flag	The controller sets the overflow flag when: • The result of a math operation results in an overflow. For example, adding 1 to a SINT generates an overflow when the value goes from 127 through -128. • The destination tag is too small to hold the value. For example, if you try to store the value 123456 to a SINT or INT tag. Use the overflow flag to verify the result of an operation is still in range. If the data being stored is a string type, S:V is set if the string is too large to fit into the destination tag. If applicable, set S:V with an OTE or OTL instruction. Select **Controller Properties > Advanced tab > Report Overflow Faults** to enable or disable reporting overflow faults. If an overflow occurs while evaluating an array subscript, a minor fault is generated and a major fault is generated to indicate the index is out of range.
S:C Carry flag	The controller sets the carry flag when the result of a math operation resulted in the generation of a carry out of the most significant bit. Only the ADD and SUB instructions, and not the + and – operators, with integer values affect this flag.
S:MINOR Minor fault flag	The controller sets the minor fault flag when there is at least one minor program fault. Use the minor fault tag to test if a minor fault occurred. This bit only triggers by programming faults, such as overflow. It is not triggered by a battery fault. The bit clears at the beginning of every scan. If applicable, explicitly set S:MINOR with an OTE or OTL instruction.

IMPORTANT: The math status flags are set based on the stored value. Instructions that normally do not affect math status flags might appear to affect math status flags if type conversion occurs from mixed data types for the instruction parameters. The type conversion process sets the math status flags.

Expressions in Array Subscripts

Expressions do not set status flags based on the results of math operations. If expressions overflow:

- A minor fault generates if the controller is configured to generate minor faults.
- A major fault (type 4, code 20) generates because the resulting value is out of range.

If an array subscript is too large (out of range), a major fault (type 4, code 20) generates.

Immediate values

When you enter an immediate value (constant) in decimal format (for example, -2, 3) the controller stores the value by using 32 bits. If you enter a value in a radix other than decimal, such as binary or hexadecimal, and do not specify all 32 bits, the controller places a zero in the bits that you do not specify (zero-fill).

IMPORTANT: Zero-fill of immediate binary, octal or hexadecimal values less than 32 bits.

If you enter	The controller stores
-1	16#ffff ffff (-1)
16#ffff (-1)	16#0000 ffff (65535)
8#1234 (668)	16#0000 029c (668)
2#1010 (10)	16#0000 000a (10)

Integer Immediate Values

If you enter	The controller stores
Without any suffix	DINT
"U"	UDINT
"L"	LINT
"UL"	ULINT

Floating Point Immediate Values

If you enter	The controller stores
Without any suffix	REAL
"L"	LREAL

Data conversions

Data conversions occur when mixing data types in programming. When programming ladder diagram, mix data types for the parameters within one instruction or expression.

Instructions execute faster and require less memory if all the operands of the instruction use:

- The same data type.
- An intermediate data type:
  - If mixing data types or use tags that are not the optimal data type, the controller converts the data according to these rules:
    - Operands are converted according to the ranking of data types from SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT, REAL, and LREAL with ranking from 1 (the lowest) to 10 (the highest).

To reduce the time and memory for converting data, use the same data type for all the operands of an instruction.

Convert SINT or INT to DINT or DINT to LINT

A SINT or INT input source tag gets promoted to a DINT value by a sign-extension for Source Tag. Instructions that convert SINT or INT values to DINT values use one of the following conversion methods.

This conversion method	Converts data by placing
Sign-extension	The value of the leftmost bit (the sign of the value) into each bit position to the left of the existing bits until there are 32 or 64 bits.
Zero-fill	Zeros to the left of the existing bits until there are 32 or 64 bits.

Logical instructions use zero fill. All other instructions use sign-extension

The following example shows the results of converting a value using sign- extension and zero-fill.

This value	2#1111_1111_1111_1111	(-1)
Converts to this value by sign-extension	2#1111_1111_1111_1111_1111_1111_1111_1111	(-1)
Converts to this value by zero-fill	2#0000_0000_0000_0000_1111_1111_1111_1111	(65535)

If you use a SINT or INT tag and an immediate value in an instruction that converts data by sign-extension, use one of these methods to handle immediate values.

Specify any immediate value in the decimal radix.

If you enter the value in a radix other than decimal, specify all 32 bits of the immediate value. To do so, enter the value of the leftmost bit into each bit position to its left until there are 32 bits.

Create a tag for each operand and use the same data type throughout the instruction. To assign a constant value, either:

Enter it into one of the tags.

Add a MOV instruction that moves the value into one of the tags.

Use a MEQ instruction to check only the required bits.

The following examples show two ways to mix an immediate value with an INT tag. Both examples check the bits of a 1771 I/O module to determine if all the bits are on. Since the input data word of a 1771 I/O module is an INT tag, it is easiest to use a 16-bit constant value.

IMPORTANT:

- When mixing an INT tag with an immediate value, since remote_rack_1:I.Data[0] is an INT tag, the value to check it against is also entered as an INT tag. When mixing an INT tag with an immediate value, since remote_rack_1:I.Data[0] is an INT tag, the value to check it against is also entered as an INT tag.
- When mixing an INT tag with an immediate value, since remote_rack_1:I.Data[0] is an INT tag, the value to check it against first moves into int_0, also an INT tag. The EQU instruction then compares both tags.

Convert Integer to REAL

The controller stores REAL values in IEEE single-precision, floating-point number format. It uses one bit for the sign of the value, 23 bits for the base value, and eight bits for the exponent (32 bits total). If you mix an integer tag (SINT, INT, or DINT) and a REAL tag as inputs in the same instruction, the controller converts the integer value to a REAL value before the instruction executes.

- A SINT or INT value always converts to the same REAL value.
- A DINT value may not convert to the same REAL value:
- A REAL value uses up to 24 bits for the base value (23 stored bits plus a 'hidden' bit).
- A DINT value uses up to 32 bits for the value (one for the sign and 31 for the value).

If the DINT value requires more than 24 significant bits, it might not convert to the same REAL value. If it will not, the controller stores the uppermost 24 bits rounded to the nearest even value.

NOTE: The Logix Designer application interprets numbers differently depending on whether the controller model is a 5x80 controller or a 5x70 controller. For example:

- For a 5x70 controller, Logix Designer interprets literal 2 as a REAL.
- For a 5x80 controller, Logix Designer interprets literal 2 as a DINT.

Convert DINT to SINT or INT

To convert a DINT value to a SINT or INT value, the controller truncates the upper portion of the DINT and stores the lower bits that fit in the data type. If the value is too large the conversion generates an overflow.

Convert a DINT to an INT and a SINT

This DINT value	Converts to this smaller value	
16#0001_0081 (65,665)	INT	16#0081 (129)
	SINT	16#81 (-127)

Convert REAL to SINT, INT, or DINT

To convert a REAL value to an integer value, the controller rounds any fractional part and stores the bits that fit in the result data type. If the value is too large the conversion generates an overflow.

Numbers round as in the following examples.

Fractions < 0.5 round down to the nearest whole number.

Fractions > 0.5 round up to the nearest whole number.

Fractions = 0.5 round up or down to the nearest even number.

Conversion of REAL values to DINT values

This REAL value	Converts to this DINT value
-2.5	-2
-3.5	-4
-1.6	.2
-1.5	.2
-1.4	.1
1.4	1
1.5	2
1.6	2
2.5	2
3.5	4

Elementary data types

The controller supports the elementary data types defined in IEC 1131-3 defined data types. The elementary data types are:

Data type	Description	Range
BOOL	1-bit boolean	0 = cleared 1 = set
SINT	1-byte integer	-128 to 127
INT	2-byte integer	-32,768 to 32,767
DINT	4-byte integer	-2,147,483,648 to 2,147,483,647
REAL	4-byte floating-point number	$-3.402823E^{38}$ to $-1.1754944E^{-38}$ (negative values) and 0 and $1.1754944E^{-38}$ to $3.402823E^{38}$ (positive values)
LINT	8-byte integer	0 to 32,535,129,599,999,999
USINT	1-byte unsigned integer	0 to 255
UINT	2-byte unsigned integer	0 to 65,535

Data type	Description	Range
UDINT	4-byte unsigned integer	0 to 4,294,967,295
ULINT	8-byte unsigned integer	0 to 18,446,744,073,709,551,615
REAL	4-byte floating-point number	-3.4028235E38 to -1.1754944E-38 (negative values) and 0.0 and 1.1754944E-38 to 3.4028235E38 (positive values)
LREAL	8-byte floating-point number	-1.7976931348623157E308 to -2.2250738585072014E-308 (negative values) and 0.0 and 2.22507385850720l4E-308 to 1.7976931348623157E308 (positive values)

The controller handles all immediate values as DINT data types.

Data type conversions

Conversion	Result		
Larger integer to smaller integer	The controller truncates the upper portion of the larger integer and generates an overflow. For example:		
	Decimal		Binary
	DINT	65,665	0000_0000_0000_0001_0000_0000_1000_0001
	INT	129	0000_0000_1000_0001
	SINT	-127	1000_0001
SINT or INT to REAL	No data precision is lost		
DINT to REAL	Data precision could be lost. Both data types store data in 32 bits, but the REAL type uses some of its 32 bits to store the exponent value. If precision is lost, the controller takes it from the least-significant portion of the DINT.		
LREAL to LREAL	No data precision is lost.		

<table>
<tr><td>LREAL TO REAL</td><td>Data precision could be lost.</td></tr>
<tr><td>LREAL/REAL to unsigned integer</td><td>Data precision could be lost. If the source value is too big to fit into destination the controller stores what it can and may produce an overflow.</td></tr>
<tr><td>Signed Integer/Unsigned Integer to LREAL/REAL</td><td>If the integer value has more significant bits than can be stored in the destination, the lower bits will be truncated.</td></tr>
<tr><td>Signed integer to unsigned integer</td><td>If the source value is too big to fit into destination, the controller stores what it can and may produce an overflow.</td></tr>
<tr><td>Unsigned integer to signed integer</td><td>If the source value is too big to fit into destination, the controller stores what it can and may produce an overflow.</td></tr>
<tr><td>REAL to integer</td><td>The controller rounds the fractional part and truncates the upper portion of the non-fractional part. If data is lost, the controller sets the overflow status flag.
Rounding is to the nearest whole number:
less than 0.5, round down; equal to 0.5, round to nearest even integer; greater than 0.5, round up
For example:
<table>
<tr><th>REAL (source)</th><th>DINT (result)</th></tr>
<tr><td>1.6</td><td>2</td></tr>
<tr><td>-1.6</td><td>-2</td></tr>
<tr><td>1.5</td><td>2</td></tr>
<tr><td>-1.5</td><td>-2</td></tr>
<tr><td>1.4</td><td>1</td></tr>
<tr><td>-1.4</td><td>-1</td></tr>
<tr><td>2.5</td><td>2</td></tr>
<tr><td>-2.5</td><td>-2</td></tr>
</table></td></tr>
</table>

Do not convert data to or from the BOOL data type.

IMPORTANT: The math status flags are set based on the value being stored. Instructions that normally do not affect math status keywords might appear to do so if type conversion occurs because of mixed data types for the instruction parameters. The type conversion process sets the math status keywords.

Safety Data Types

The Logix Designer application prevents the modification of a User Defined or Add-On Defined type that would cause an invalid data type for User Defined or Add-On Defined types that are referenced directly or indirectly by a Safety tag. (This includes nested structures.)

Safety tags can be composed of the following data types:

- All elementary data types.
- Predefined types that are used for safety application instructions.
- User-defined data types or arrays that are composed of the previous two types.

Online edits of user-defined data type member names in safety tags

Online editing is allowed for member names of user-defined data types on CompactLogix 5380, Compact GuardLogix 5380, ControlLogix 5580, GuardLogix 5580, and ControlLogix 5590 controllers. However, online editing is disabled when a user-defined data type is used on a safety tag and the controller is in the Safety Secured state.

Related information

Math status flags on page 591

Floating Point Values

Logix controllers handle floating point values according to the IEEE 754 standard for floating-point arithmetic. This standard defines how floating point numbers are stored and calculated. The IEEE 754 standard for floating point math was designed to provide speed and the ability to handle very large numbers in a reasonable amount of storage space.

A REAL tag stores a single-precision, normalized floating-point number.

An LREAL tag stores a double-precision, normalized floating-point number.

The controllers support these elementary data types:

- REAL
- LREAL

Denormalized numbers and -0.0 are treated as 0.0

If a computation results in a NAN value, the sign bit could be positive or negative. In this situation, the software displays 1#.NAN with no sign.

Not all decimal values can be exactly represented in this standard format, which results in a loss of precision. For example, if you subtract 10 from 10.1, you expect the result to be 0.1. In a Logix controller, the result could very well be 0.10000038. In this example, the difference between 0.1 and 0.10000038 is .000038%, or practically zero. For most operations, this small inaccuracy is insignificant. To put things in perspective, if you were sending a floating point value to an analog output module, there would be no difference in the output voltage for a value being sent to the module that differs by .000038%.

Guidelines for Floating-point Math Operations

Follow these guidelines:

When performing certain floating-point math operations, there may be a loss of precision due to rounding error. Floating-point processors have their own internal precision that can impact resultant values.

Do not use floating point math for money values or for totalizer functions. Use INT or DINT values, scale the values up, and keep track of the decimal place (or use one INT or DINT value for dollars, and a second INT or DINT value for cents).

Do not compare floating-point numbers. Instead, check for values within a range. The LIMIT instruction is provided specifically for this purpose.

Totalizer Examples

The precision of the REAL data type affects totalization applications such that errors occur when adding very small numbers to very large numbers.

For example, add 1 to a number over a period of time. At some point the add will no longer affect the result because the running sum is much greater than 1, and there are not enough bits to store the entire result. The add stores as many upper bits as possible and discards the remaining lower bits.

To work around this, do math on small numbers until the results get large. Then, transfer them to another location for additional large-number math. For example:

- x is the small incremented variable.
- y is the large incremented variable.
- z is the total current count that can be used anywhere.
- x = x+1;
- if x = 100,000;
- {
- y = y + 100,000;
- x = 0;
- }
- z = y + x;

Or another example:

- x = x + some_tiny_number;
- if (x >= 100)
- {
- z = z + 100;
- x = x - 100; // there might be a tiny remainder
- }

Index through arrays

To dynamically change the array element that your logic references, use tag or expression as the subscript to point to the element. This is similar to indirect addressing in PLC-5 logic. Use these operators in an expression to specify an array subscript:

Operator	Description
+	add
-	subtract/negate
*	multiply

Operator	Description
/	divide
AND	AND
FRD	BCD to integer
NOT	complement
OR	OR
TOD	integer to BCD
SQR	square root
XOR	exclusive OR

For example:

Definitions	Example	Description
my_list defined as DINT[10]	my_list[5]	This example references element 5 in the array. The reference is static because the subscript value remains constant.
my_list defined as DINT[10] position defined as DINT	MOV the value 5 into position my_list[position]	This example references element 5 in the array. The reference is dynamic because the logic can change the subscript by changing the value of position.
my_list defined as DINT[10] position defined as DINT offset defined as DINT	MOV the value 2 into position MOV the value 5 into offset my_list[position+offset]	This example references element 7 (2+5) in the array. The reference is dynamic because the logic can change the subscript by changing the value of position or offset.

Make sure any array subscript you enter is within the boundaries of the specified array. Instructions that view arrays as a collection of elements generate a major fault (type 4, code 20) if a subscript exceeds its corresponding dimension.

Bit Addressing

Bit addressing is used access a particular bit within a larger container. Larger containers include any integer, structure or BOOL array. For example:

Definition	Example	Description
Variable0 defined as LINT has 64 bits	variable0.42	This example references the bit 42 of variable0.
variable1 defined as DINT	variable1.2	This example references the bit 2 of variable1.

Definition	Example	Description
has 32 bits		
variable2 defined as INT has 16 bits	variable2.15	This example references the bit 15 of variable2.
variable3 defined as SINT holds 8 bits	variable3.[4]	This example references bit 4 of variable3.
variable4 defined as COUNTER structure has 5 status bits	variable4.DN	This example references the DN bit of variable4.
MyVariable defined as BOOL[100] MyIndex defined as SINT	MyVariable[(MyIndex AND NOT 7) / 8].[MyIndex AND 7]	This example references a bit within a BOOL array.
MyArray defined as BOOL[20]	MyArray[3]	This example references the bit 3 of MyArray.
variable5 defined as ULINT holds 64 bits	variable5.53	This example references the bit 53 of variable5.

Use Bit Addressing anywhere a BOOL typed tag is allowed.

Related information

Index through arrays on page 600

Rockwell Automation Support

Use these resources to access support information.

Technical Support Center	Find help with how-to videos, FAQs, chat, user forums, and product notification updates.	rok.auto/support
Local Technical Support Phone Numbers	Locate the telephone number for your country.	rok.auto/phonesupport
Technical Documentation Center	Quickly access and download technical specifications, installation instructions, and user manuals.	rok.auto/techdocs
Literature Library	Find installation instructions, manuals, brochures, and technical data publications.	rok.auto/literature
Product Compatibility and Download Center (PCDC)	Get help determining how products interact, check features and capabilities, and find associated firmware.	rok.auto/pcdc

Documentation Feedback

Your comments help us serve your documentation needs better. If you have any suggestions on how to improve our content, complete the form at rok.auto/docfeedback.

Waste Electrical and Electronic Equipment (WEEE)

At the end of life, this equipment should be collected separately from any unsorted municipal waste.

Rockwell Automation maintains current product environmental information on its website at rok.auto/pec.

Allen-Bradley, expanding human possibility, and Rockwell Automation are trademarks of Rockwell Automation, Inc.

Trademarks not belonging to Rockwell Automation are property of their respective companies.

Rockwell Otomasyon Ticaret A.Ş. Kar Plaza İş Merkezi E Blok Kat:6 34752 İçerenköy, İstanbul, Tel: +90 (216) 5698400 EEE Yönetmeliğine Uygundur

Connect with us.

rockwellautomation.com — expanding **human possibility**®

AMERICAS: Rockwell Automation, 1201 South Second Street, Milwaukee, WI 53204-2496 USA, Tel: (1) 414.382.2000
EUROPE/MIDDLE EAST/AFRICA: Rockwell Automation NV, Pegasus Park, De Kleetlaan 12a, 1831 Diegem, Belgium, Tel: (32) 2663 0600
ASIA PACIFIC: Rockwell Automation SEA Pte Ltd, 2 Corporation Road, #04-05, Main Lobby, Corporation Place, Singapore 618494, Tel: (65) 6510 6608
UNITED KINGDOM: Rockwell Automation Ltd., Pitfield, Kiln Farm, Milton Keynes, MK11 3DR, United Kingdom, Tel: (44)(1908) 838-800

Publication 1756-RM006P-EN-P - September 2025
Supersedes Publication 1756-RM0060-EN-P - September 2024

Copyright © 2025, Rockwell Automation, Inc. All rights reserved. Printed in the U.S.A.