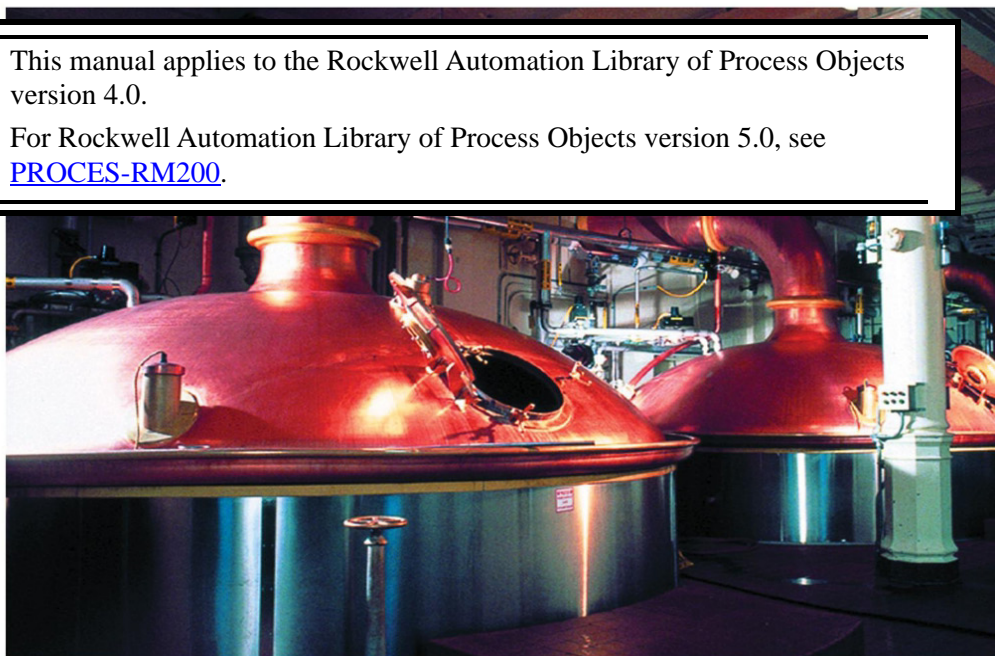
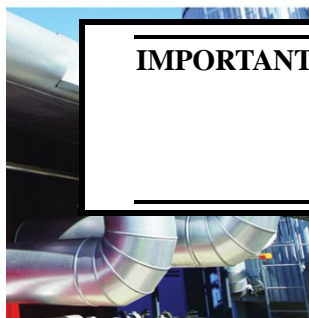


Rockwell Automation Library of Process Objects: Logic Instructions

Version 4.10.06

IMPORTANT This manual applies to the Rockwell Automation Library of Process Objects version 4.0.
For Rockwell Automation Library of Process Objects version 5.0, see [PROCES-RM200](#).



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.

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).

Preface	Summary of Changes	12
	Access the Attachments	12
	Open Content	12
	How to Use Attachments	13
	Additional Resources	14
	 Chapter 1	
I/O Processing	Purpose	15
	Basic Analog Input (P_AIn)	21
	Functional Description	22
	Required Files	22
	Operations	22
	Simulation	24
	Programming Example	24
	Analog Input Channel (P_AIChan)	26
	Required Files	27
	Operations	27
	Programming Example	29
	Advanced Analog Input (P_AInAdv)	30
	Functional Description	30
	Required Files	31
	Operations	32
	Programming Example	34
	Dual Sensor Analog Input (P_AInDual)	35
	Functional Description	35
	Required Files	36
	Operations	37
	Programming Example	38
	Multiple Analog Input (P_AInMulti)	39
	Functional Description	39
	Required Files	40
	Operations	41
	Programming Example	43
	Discrete Input Object (P_DIn)	46
	Functional Description	47
	Required Files	48
	Operations	49
	Programming Example	50
	Discrete Input Object Advanced (P_DInAdv)	53
	Functional Description	53
	Required Files	54
	Operations	55
	Programming Example	56
	Discrete Output (P_DOut)	58
	Functional Description	58
	Required Files	59
	Operations	59

Programming Example.....	61
Analog Output (P_AOut)	63
Functional Description	65
Required Files.....	65
Operations.....	66
Programming Example.....	67
Pressure/Temp. Compensated Flow (P_PTComp)	68
Functional Description	68
Required Files.....	68
Operations.....	69
Programming Example.....	69
Tank Strapping Table (P_StrapTbl)	72
Functional Description	72
Required Files.....	72
Operations.....	72
Programming Example.....	74
HART Analog Input (P_AInHART)	76
HART Analog Output (P_AOutHART).....	76

Chapter 2

Regulatory and Procedural Control

Purpose	77
Proportional + Integral +Derivative Enhanced (P_PIDE)	79
Functional Description	79
Autotune	80
Required Files.....	80
Operations.....	80
Programming Example.....	82
Analog Fanout (P_Fanout)	84
Functional Description	84
Required Files.....	85
Operations.....	86
Programming Example.....	87
High or Low Selector (P_HiLoSel)	89
Required Files.....	89
Operations.....	90
Programming Example.....	91
Deadband Controller (P_DBC).....	92
Functional Description	92
Required Files.....	92
Operations.....	93
Sequencer Object (P_Seq)	96
Functional Description	96
Required Files.....	96
Step User-defined Data Type.....	97
Operator Prompt.....	97
Operations.....	97
Dosing (P_Dose).....	99

Functional Description	101
Required Files	102
Operations	102
Programming Example	105
Lead/Lag/Standby Motor Group (P_LLS)	107
Functional Description	107
Required Files	109
Operations	114
Motor Sort Algorithm	117

Chapter 3

Motors

Purpose	119
Single-speed Motor (P_Motor)	127
Functional Description	127
Required Files	127
Operations	129
Programming Example	130
Two-speed Motor (P_Motor2Spd)	132
Functional Description	132
Required Files	132
Operations	133
Programming Example	135
Reversing Motor (P_MotorRev)	136
Functional Description	136
Required Files	136
Operations	137
Programming Example	139
Hand-operated Motor (P_MotorHO)	140
Functional Description	140
Required Files	140
Operations	141
Programming Example	142
PowerFlex 523/525 Drives (P_PF52x)	143
Functional Description	143
Required Files	143
Required Drive Configuration	144
InOut Structure	145
Operations	146
Programming Example	148
PowerFlex 753 Drive (P_PF753)	158
Functional Description	158
Required Files	159
InOut Structure	160
Operations	161
Programming Example	164
PowerFlex 755, PowerFlex 755TL/TR Drives (P_PF755)	174
Functional Description	174

Required Files.....	175
Required Drive Configuration.....	176
InOut Structure.....	177
Operations.....	179
Programming Example.....	181
PowerFlex 6000 Drive (P_PF6000).....	192
Functional Description	192
Required Files.....	193
InOut Structure.....	194
Operations.....	195
Programming Example.....	198
PowerFlex 7000 Drive (P_PF7000).....	204
Functional Description	204
Required Files.....	205
InOut Structure.....	206
Operations.....	212
Programming Example.....	214
SMC-50 Smart Motor Controller (P_SMC50).....	224
Functional Description	224
Required Files.....	224
Required SMC-50 Configuration.....	225
InOut Structure.....	225
Operations.....	226
Programming Example.....	229
SMC Flex Smart Motor Controller (P_SMCFlex)	230
Functional Description	230
Required Files.....	230
Required SMC Flex Configuration	231
SMC Flex Smart Motor Controller InOut Structure	231
Operations.....	232
Programming Example.....	235
Variable-speed Drive (P_VSD).....	236
Functional Description	236
Required Files.....	237
Required Connections for a Hardwired Drive.....	238
InOut Structure.....	240
Operations.....	241
Programming Example.....	244
E1 Plus Electronic Overload Relay (P_E1PlusE)	252
Functional Description	252
Required Files.....	252
InOut Structure.....	253
Operations.....	253
Programming Example.....	254
E3/E3Plus Overload Relay (P_E3Ovld).....	257
Functional Description	257
Required Files.....	257

Required Overload Configuration	258
InOut Structure.....	258
Operations.....	259
Programming Example.....	260
E300 Electronic Overload Relay (P_E300Ovld)	262
Functional Description	262
Required Files.....	262
Required Overload Configuration	263
InOut Structure.....	264
Operations.....	265
Programming Example.....	267
Runtime and Start Counter (P_RunTime)	269
Functional Description	269
Required Files.....	270
Operations.....	271
Implementation Using EnableIn False Feature	271
Restart Inhibit for Large Motor (P_ResInh)	273
Functional Description	273
Required Files.....	274
Operations.....	274
Implementation by Using EnableIn False Feature	275

Chapter 4

Valves

Purpose	279
Analog/Pulsed Control Valve (P_ValveC)	283
Functional Description	283
Required Files.....	284
Operations.....	286
Programming Examples	287
Example 1: Manual Loading Station	287
Example 2: Ratcheting Control Valve	288
Hand-operated Valve (P_ValveHO)	291
Functional Description	291
Required Files.....	292
Operations.....	292
Programming Example.....	294
Motor-operated Valve (P_ValveMO)	295
Functional Description	295
Required Files.....	296
Operations.....	296
Programming Example.....	298
Mix-proof Valve (P_ValveMP)	300
Functional Description	300
Required Files.....	301
Operations.....	301
Programming Example.....	304
Advanced Mix-Proof Valve (P_ValveMPAdv)	306

Functional Description	307
Required Files	308
Operations	308
Programming Example	310
Solenoid-operated Valve (P_ValveSO)	314
Functional Description	314
Required Files	315
Operations	315
Programming Example	318
2-state Valve Statistics (P_ValveStats)	319
Functional Description	319
Required Files	319
Operations	320
Programming Example	321
n-Position Device (P_nPos)	323
Functional Description	324
Required Files	325
Operations	326
Programming Example	329

Chapter 5

Cross Functional

Purpose	331
Condition Gate Delay (P_Gate)	336
Functional Description	336
Required Files	336
Operations	337
Interlocks with First Out and Bypass (P_Intlk)	340
Functional Description	341
Required Files	342
Operations	342
Programming Example	343
Interlocks with First Out and Bypass - Advanced (P_IntlkAdv) ..	346
Functional Description	346
Required Files	348
Operations	348
Programming Example	350
Permissives with Bypass (P_Perm)	353
Functional Description	353
Required Files	354
Operations	354
Programming Example	355
Discrete 2-, 3-, or 4-state Device (P_D4SD)	359
Functional Description	359
Required Files	360
Operations	360
Programming Example	363

Central Reset (P_Reset)	367
Functional Description	367
Required Files	367
Operations	368
Implementation by Using EnableIn False Feature	369
Common Alarm Block (P_Alarm)	371
Functional Description	372
Required Files	374
Operations	374
Standalone Versus Embedded in Other Add-On Instructions	376
Implementation by Using the EnableIn False Feature	377
Command Source (P_CmdSrc)	378
Required Files	378
Operations	378
Associated Tags	382
Hand Command Source Example	386
External Command Source Example	387
Operator Prompt (P_Prompt)	389
Functional Description	389
Required Files	390
Controller File	390
Operations	390
Boolean Logic with Snapshot (P_Logic)	391
Functional Description	391
Required Files	392
Operations	393
Programming Example	394

Appendix A

Command Sources	397
Simulation	399

Appendix B

Long Integer and Time Instructions	401
Time and Date Instructions	405

Appendix C

Process Strategies	417
--------------------------	-----

Index	429
--------------------	------------

Command Source, Simulation Types

Additional Add-on Instructions

Process Strategies

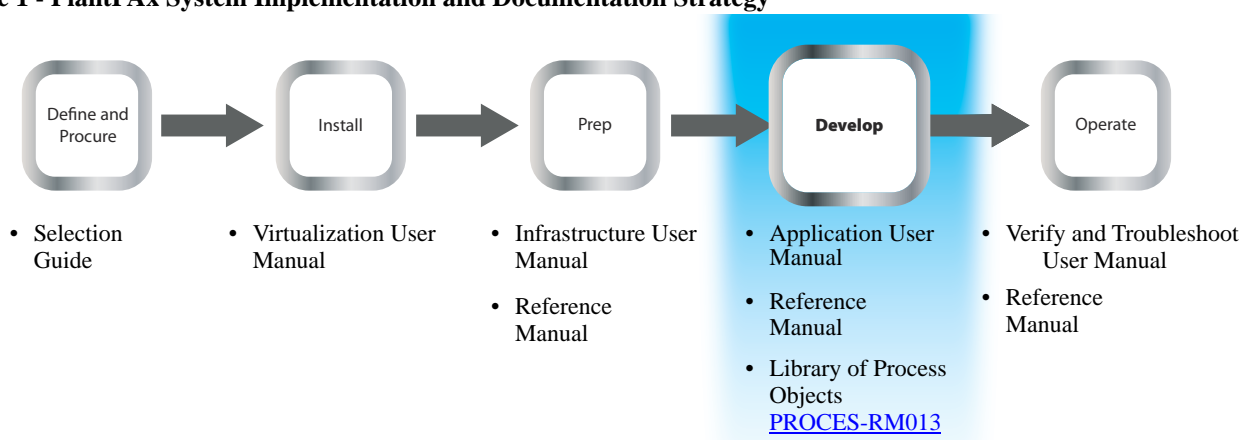
Notes:

This document helps you select the logic that is contained in Add-On Instructions that comprise the volume of Rockwell Automation® Library of Process Objects. Sections are divided into classifications that are based on what criteria the Add-On Instructions control and monitor, such as motors. Each section features a table with a brief description of each Add-On Instruction and when to use and when not to use the instruction for your project.

This document is for the operation of the Add-on Instructions. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments.

The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

Figure 1 - PlantPAx System Implementation and Documentation Strategy



- **Define and Procure** – Helps you understand the elements of the PlantPAx® system to make sure that you buy the proper components.
- **Install** – Provides direction on how to install the PlantPAx system.
- **Prep** – Provides guidance on how to get started before you develop your application.
- **Develop** – Describes the actions and libraries necessary to construct your application that resides on the PlantPAx system.
- **Operate** – Provides guidance on how to verify and maintain your systems for operation of your plant.

Summary of Changes

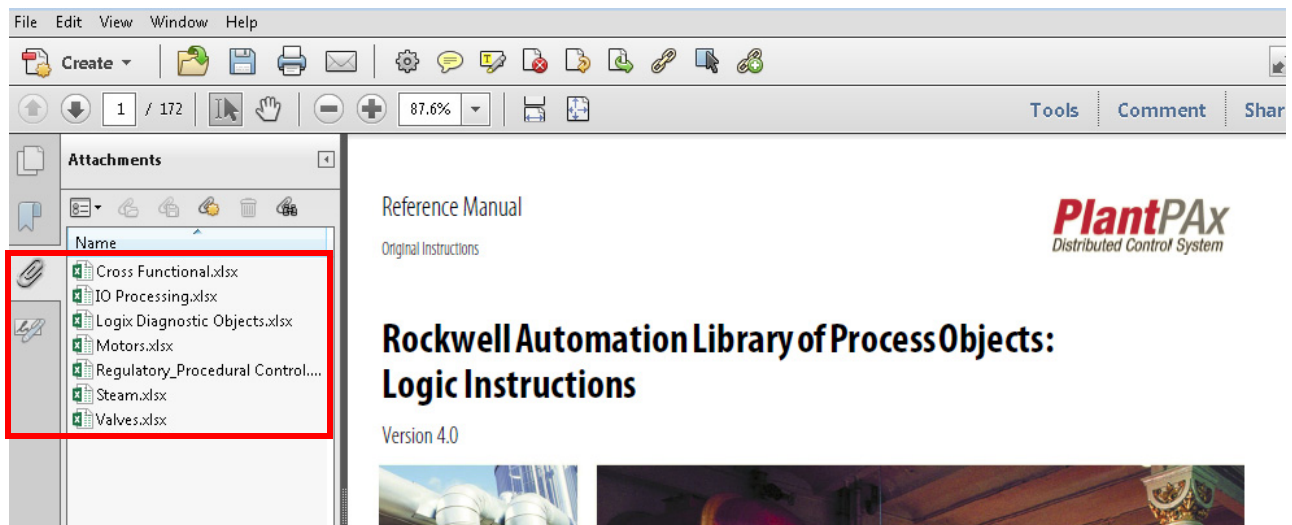
Updated P_Intlk and P_Perm Add-On Instructions

Updated attached excel sheets to reflect the parameter changes.

Access the Attachments

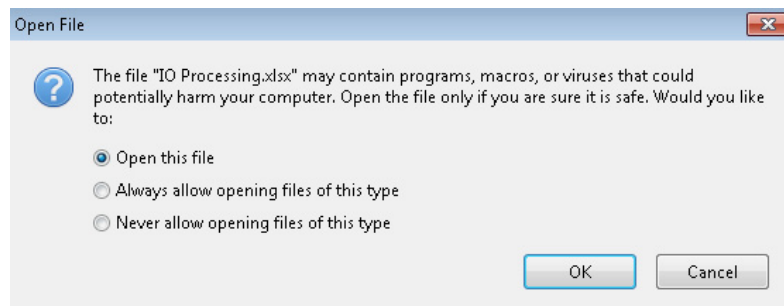
The Microsoft Excel spreadsheets that are attached to this PDF file contain input/output parameters and local configuration tags for Add-On Instructions.

To use a Microsoft Excel file, click the Attachments link (the paper clip) and double-click the desired file..



Open Content

As a precaution when you open programs or files, select one of the choices and click OK.



How to Use Attachments

Each Microsoft Excel spreadsheet has three tabs: Input parameters, Output parameters, Local configuration tags. Click the respective tab and use the search functionality to locate a parameter.

The screenshot shows the Microsoft Excel interface with the 'Input Parameters' tab selected. The formula bar displays 'IOFault.Cfg_Exists'. The spreadsheet contains the following data:

	A	B	C	D	E	F
	Library Object	Input Parameter	Data Type	Alias For	Default	Description
1						
1159	P_SMC50	Cfg_OvrPermIntlk	BOOL		0	1 = Override ignores bypassable permissive/interlock. 0 = Always use permissive/interlock.
1160	P_SMC50	Cfg_ShedOnFailToStart	BOOL		0	1 = Stop Motor and alarm on Fail to Start. 0 = Alarm only on Fail to Start.
1161	P_SMC50	Cfg_ShedOnIOFault	BOOL		0	1 = Stop Motor and alarm on I/O Fault. 0 = Alarm only on I/O Fault.
1162	P_SMC50	Cfg_HasFailToStartAlm	BOOL	FailToStart.Cfg_Exists	0	1 = Fail to Start alarm, Fail to Stop alarm, Interlock Trip alarm, Motor Fault alarm, or I/O Fault alarm exists and is checked.
1163		Cfg_HasFailToStopAlm		FailToStop.Cfg_Exists		
1164		Cfg_HasIntlkTripAlm		IntlkTrip.Cfg_Exists		
1165		Cfg_HasMotorFaultAlm		MotorFault.Cfg_Exists		
1166		Cfg_ShedOnIOFault		IOFault.Cfg_Exists		
1167	P_SMC50	Cfg_FailToStartResetReqd	BOOL	FailToStart.Cfg_ResetReqd	0	1 = Reset required to clear Fail to Start alarm, Fail to Stop alarm, Interlock Trip alarm, Motor Fault alarm, or I/O Fault alarm.
1168		Cfg_FailToStopResetReqd		FailToStop.Cfg_ResetReqd		
1169		Cfg_IntlkTripResetReqd		IntlkTrip.Cfg_ResetReqd		
1170		Cfg_MotorFaultResetReqd		MotorFault.Cfg_ResetReqd		
1171		Cfg_IOFaultResetReqd		IOFault.Cfg_ResetReqd		
1172	P_SMC50	Cfg_FailToStartAckReqd	BOOL	FailToStart.Cfg_AckReqd	1	1 = Acknowledge required for Fail to Start alarm, Fail to Stop alarm, Interlock Trip alarm, Motor Fault alarm, or I/O Fault alarm.

The 'Input Parameters' tab is highlighted at the bottom of the spreadsheet.

Additional Resources

These documents contain additional information that concerns related products from Rockwell Automation.

Resource	Description
PlantPAx Distributed Control System Selection Guide, publication PROCES-SG001	Provides basic definitions of system elements and sizing guidelines for procuring a PlantPAx system.
PlantPAx Distributed Control System Infrastructure Configuration User Manual, publication PROCES-UM001	Describes procedures for how to configure system components that comprise a PlantPAx modern DCS.
PlantPAx Distributed Control System Application Configuration User Manual, publication PROCES-UM003	Describes procedures to start development of your PlantPAx distributed control system.
PlantPAx Distributed Control System Reference Manual, publication PROCES-RM001	Provides characterized recommendations for implementing your PlantPAx system.
Rockwell Automation Library of Process Objects Reference Manuals: publication PROCES-RM013 Publication PROCES-RM014	Provides an overview of the code objects, display elements, and faceplates that comprise the Rockwell Automation Library of Process Objects.
Rockwell Automation Library of Logix Diagnostic Objects Reference Manual, publication PROCES-RM003	Provides information on Add-On Instructions that monitor Logix controllers to diagnose issues that include memory usage, communication, and control.
Rockwell Automation Library of Steam Table Instructions, publication PROCES-RM004	Provides Add-On Instructions for to calculate temperature and pressure steam tables.
Redundant I/O System User Manual, publication 1715-UM001	Explains how to install and configure the 1715 Redundant I/O system.
Product Compatibility and Download Center at http://www.rockwellautomation.com/rockwellautomation/support/pcdc.page	Website helps you find product-related downloads including firmware, release notes, associated software, drivers, tools, and utilities.
Rockwell Automation Sample Code website at http://samplecode.rockwellautomation.com	Accesses a Rockwell Automation webpage to search for sample code.

You can view or download publications at <http://www.rockwellautomation.com/literature/>. To order paper copies of technical documentation, contact your local Allen-Bradley distributor or Rockwell Automation sales representative.

I/O Processing

Purpose

This chapter is for the operation of the Add-on Instructions. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The Process Objects in this group provide analog and discrete input/output signal processing. Pressure/temperature compensated flow calculations and cylindrical tank level interpolations are also provided.

[Table 1](#) describes the objects in this group, including when to use and not to use each one.

Table 1 - I/O Processing Objects

Process Object	Description	When to Use	When Not to Use
Basic Analog Input (P_AIn)	<p>This instruction monitors one analog value, typically from a channel of an analog input module, and provides alarms when the analog value exceeds user-specified thresholds (high and low).</p> <p>The instruction also provides capabilities for linear scaling of an analog input value from raw (input) units to engineering (output) units. The logic also provides entries of a substitute Process Variable, providing handling of an out-of-range or faulted input.</p> <p>To keep the instruction memory and execution footprint small, certain capabilities, used less frequently, are reserved for the P_AInAdv Add-On Instruction.</p>	<ul style="list-style-type: none"> • Display a temperature, flow, pressure, level, or other signal from one field instrument on your HMI. • To scale, alarm, or use HMI features for a single analog Input, or any analog (quantity) value. <ul style="list-style-type: none"> – Linear scaling from raw to engineering units – High, Low, High-High, Low-Low, and Out of Range alarms (with deadband, on-delay and off-delay per alarm) – Indicator graphic object with label and engineering units – Faceplate with status threshold entry, and maintenance capability for substitute PV 	<ul style="list-style-type: none"> • Analog input signal is handled by another instruction. For example, the Speed Feedback for a variable speed drive is handled by the P_VSD instruction. It is not necessary to use the P_AIn Add-On Instruction first. Wire or map the input directly to P_VSD. • To display only a number on a screen and do not need any of the scaling or alarming features. Use a numeric display field instead. • Need advanced capabilities, such as square root extraction (for example, orifice flowmeters), rate-of-change alarming or limiting, or alarming for deviation from a reference value. Use the P_AInAdv instruction. • There are dual sensors for one process variable, such as dual PH meters, and one or the other sensor (or their average) is selected. Use the P_AInDual instruction. • There are more than two sensors for one process variable and need to use the average or median sensor value. Use the P_AInMulti instruction.

Table 1 - I/O Processing Objects

Process Object	Description	When to Use	When Not to Use
Analog Input Channel (P_AIChan)	<p>This instruction monitors one analog input channel and provides one configurable alarm.</p> <p>This instruction is associated with other instructions, with one instance being used for each analog input of the associated instruction.</p> <p>The P_AIChan faceplate is called from other faceplates, such as the associated analog input instruction faceplate.</p>	<p>The P_AIChan Add-On Instruction monitors an input channel for the following conditions:</p> <ul style="list-style-type: none"> Invalid configuration I/O module fault Input out of range Instrument reports the following conditions: <ul style="list-style-type: none"> Out of specification (uncertain) Function check (substitute PV entered manually) Maintenance required Channel fault Input not-a-number (floating point exception) Input stuck (unchanging) <p>For each condition, the Process Variable (PV) quality to report can be configured as follows:</p> <ul style="list-style-type: none"> Good Uncertain Bad (raises Fail alarm) <p>For each condition, the following actions can be taken:</p> <ul style="list-style-type: none"> Pass the PV through unchanged Apply a configured replacement PV value Use the last good PV value 	
Advanced Analog Input (P_AInAdv)	<p>This instruction monitors one analog value, typically from an analog input I/O module.</p> <p>The P_AInAdv instruction has the following advanced features that are not included in the basic analog input:</p> <ul style="list-style-type: none"> Square root characterized scaling of the input value from raw (input card) units to engineering (display) units. Square root characterized scaling is used with orifice plates or other pressure-differential elements for flow measurement when the transmitter does not provide square root characterization. The square root scaling in the P_AInAdv instruction works with \pm pressure differential to provide positive or negative flow values. Operator or Program entry of a reference (setpoint) value with configurable alarms for PV deviation above or below the reference value. Calculation of the PV rate of change and configurable high rate of change alarming. 	<ul style="list-style-type: none"> Display a temperature, flow, pressure, level, or other signal from a field instrument on your HMI. Need any of the advanced features: square root characterization, deviation display/alarms, or rate of change display/alarms. 	<ul style="list-style-type: none"> Need only basic analog input features and don't need any of the advanced features. Use the P_AIn Analog Input instruction instead; it uses less memory and is processed faster. Need only to display a number on a screen and do not need any scaling, alarming, or faceplate features. Use an HMI numeric display object instead. Have dual sensors for a single process variable (for example, dual PH probes and transmitters) and need to choose one sensor, the other, or their average. Use the P_AInDual instruction instead. Have more than two sensors for one process variable and need to use the average or median sensor value. Use the P_AInMulti instruction instead.

Table 1 - I/O Processing Objects

Process Object	Description	When to Use	When Not to Use
Dual Sensor Analog Input (P_AInDual)	<p>This instruction monitors one analog Process Variable (PV) by using two analog input signals (dual sensors, dual transmitters, and dual-input channels).</p> <p>This instruction has the following advanced features that are not included in the basic analog input:</p> <ul style="list-style-type: none"> • Dual-inputs • Alarm if difference between the two input PVs exceeds a configured limit 	<ul style="list-style-type: none"> • Display a temperature, flow, pressure, level, or other PV on your HMI. • Have dual sensors for the single process variable (for example, dual PH probes and transmitters) and need to select one sensor, the other, or their average, minimum, or maximum. 	<ul style="list-style-type: none"> • Have more than two sensors for one process variable and need to use the average or median sensor value. Use the P_AInMulti instruction instead. • Need only display a number on a screen and do not need any scaling, alarming, or faceplate features. You can use an HMI numeric display object. • Display a PV from a single sensor, transmitter, and input. For basic scaling and High, High-High, Low, and Low-Low threshold alarms, use the P_AIn instruction. • Need additional features, including square root signal characterization, rate of change display or alarming, or reference entry with deviation alarming, use the P_AInAdv instruction.
Multiple Analog Input (P_AInMulti)	<p>This instruction monitors one analog process variable (PV) by using up to eight analog input signals (sensors, transmitters, input channels).</p> <p>The P_AInMulti instruction displays a temperature or other process variable by averaging multiple instruments.</p>	<ul style="list-style-type: none"> • Display a temperature, pressure, level, or other PV on your HMI or use the PV in your control logic, and the following applies: <ul style="list-style-type: none"> – Three or more sensors for that PV (for example, six thermocouples and six thermocouple input channels on an I/O card). – Want the PV to be the mean or median of the input values of the sensors. 	<ul style="list-style-type: none"> • Have only two sensors for the PV. Use the P_AInDual instruction instead. The P_AInDual instruction lets you select Sensor A, Sensor B, the greater (high select), the lesser (low select), or the average of the two sensors. The P_AInDual instruction also provides automatic failover to the good sensor when one sensor fails. • Have multiple PVs, such as six temperatures, when each has its own significance. Use multiple instances of the P_AIn instruction instead.
Discrete Input Object (P_DIIn)	<p>This instruction receives and processes a single discrete condition (the PV), typically for a channel of a discrete input card. The instruction is used with any discrete (BOOL) signal.</p>	<ul style="list-style-type: none"> • Want to display the state of a process temperature, level, flow, proximity, pressure, or other switch. • Need any of these signal processing or alarming features for a Discrete input or any Discrete (bit) value: <ul style="list-style-type: none"> – De-bounce of the discrete input signal. – Target Disagree status and optional alarm when the Discrete input is not in a Target state for some period and a gating condition is true for some time. – Display of the input state with configurable text on an HMI object. – Ability for maintenance personnel to provide a substitute value when the device has failed. 	<ul style="list-style-type: none"> • Need only to show or not show the state of a bit on an HMI display. Use basic display objects (text, multi-state indicators) with appropriate animation instead. • Need only to generate an alarm from some condition you already have in your code. Use the P_Alarm instruction or the ALMD built-in instruction instead.

Table 1 - I/O Processing Objects

Process Object	Description	When to Use	When Not to Use
Discrete Input Object Advanced (P_DInAdv)	This instruction receives and processes a single discrete condition (the PV), typically for a channel of a discrete input card. The instruction is used with any discrete (BOOL) signal. The instruction provides additional capability for legacy equipment to handle dual abnormal situation indication and speed switch capabilities. When configured for speed switch functionality this instruction receives and processes a motor run feedback (BOOL) signal for speed switch functionality and can also receive a variable speed drive speed feedback (0-100%) value.	<ul style="list-style-type: none"> • Need any of these signal processing or alarming features for a Discrete input or any Discrete (bit) value: <ul style="list-style-type: none"> – Dual abnormal situation indication. Used for sequential indication of a warning condition followed by a failure condition – Separate time configurations with individual messages and indications – Variable speed drive auto adjust of warning and fail delay times according to speed feedback – Speed switch functionality – De-bounce of the discrete input signal. – Target Disagree status and optional alarm when the Discrete input is not in a Target state for some period and a gating condition is true for some time. – Display of the input state with configurable text on an HMI object. – Ability for maintenance personnel to provide a substitute value when the device has failed. 	<p>Need only to show or not show the state of a bit on an HMI display. Use basic display objects (text, multi-state indicators) with appropriate animation instead.</p> <ul style="list-style-type: none"> • Need only to generate an alarm from some condition you already have in your code. Use the P_Alarm instruction or the ALMD built-in instruction instead. • Don't need dual abnormal situation indication. Use the P_DIn instruction instead. • Don't need speed switch functionality. Use the P_DIn instruction instead.
Discrete Output (P_DOut)	This instruction controls a device by a single discrete output signal and optionally monitors feedback from the device to check for device failures. The P_DOut instruction operates in a variety of modes, and can provide steady, single pulsed, or continually pulsed output.	<ul style="list-style-type: none"> • Need to operate a device by using a single discrete output and that device is not supported by other Rockwell Automation Library of Process Objects Add-On Instructions. • Have a device, such as a valve or motor, that is supported by other Add-On Instructions. You want the device to use non-standard state names, such as 'recycle' and 'deliver' for a diverter valve, rather than the fixed names used in the other instruction, such as 'closed' and 'open'. The P_DOut instruction has configurable names for each of the device states. • Need to operate a device that requires pulsing (single-pulse or continuous). The P_DOut instruction provides on-delay timing, off-delay timing, and commands for single On pulse, single Off pulse, and continuous pulse stream. The instruction also provides On and Off. For example, the P_DOut instruction can be a good choice for pilot lights or stack lights that require blinking. 	<ul style="list-style-type: none"> • Need to operate a device that has more than one discrete output or more than two discrete inputs for device feedback. See the P_D4SD or P_nPos instructions. • Need to operate a single-speed motor, solenoid valve, or other device that is better supported by other Rockwell Automation Library of Process Objects Add-On Instructions. P_Motor and P_ValveSO instructions, for example, more closely model the device under control and can provide better diagnostics for the device. • Need to operate a continuously variable device. Use the P_AOut, P_ValveC or P_VSD instruction instead.

Table 1 - I/O Processing Objects

Process Object	Description	When to Use	When Not to Use
Analog Output (P_AOut)	<p>This instruction manipulates an analog output to control a field device, such as a control valve or a motorized gate positioner. The output responds to an Operator (manual) or Program setting of the Controlled Variable (CV) signal.</p> <p>The P_AOut instruction controls the analog output in a variety of modes (Operator, Program, override, Maintenance, Hand), monitoring for fault conditions.</p>	Use this instruction when you need the functionality of a Manual Loading Station to generate an open-loop CV signal for a final control element or any analog output signal.	<ul style="list-style-type: none"> Want closed-loop control of a process variable. The P_AOut instruction provides only open-loop generation of an analog controlled variable. There are a number of instructions available that provide closed-loop control algorithms. They include functions necessary to tie directly to the analog output and/or outputs, including auto/manual CV selection and output scaling. Have another instruction that works directly with one or more analog outputs. For example, the P_VSD instruction has the functionality that is needed for its speed reference and output-datalink analog outputs. Use the outputs of that instruction tied directly to the analog outputs. No P_AOut instruction is required. Have a valve with position feedback. Use the P_ValveC instruction instead.
Pressure/Temp. Compensated Flow (P_PTComp)	<p>This instruction calculates a flow at standard temperature and pressure, essentially a mass flow rate given a volumetric flow rate or differential pressure measurement. This instruction also requires measurements of the actual temperature and pressure of the flowing gas.</p> <p>The P_PTComp instruction is intended as a calculation function only, between other blocks, and no HMI components are provided. If a faceplate and/or alarms are needed, the calculated output from the instruction can be sent to a P_AIn instruction for alarming and display.</p>	<ul style="list-style-type: none"> Need to measure the flow of a gas, or the differential pressure of a gas that behaves (approximately) as an Ideal Gas across a flow element. For example, an orifice plate. Need to measure the temperature and pressure of the gas. Want to express the flow as a Flow at Standard Temperature and Pressure (STP), such as Standard Cubic Feet per Minute (SCFM). 	<ul style="list-style-type: none"> Need to measure the flow of steam by using methods that are defined by ASME (or other standard or authority having jurisdiction), including the use of steam table. The measurement determines the properties of the steam to calculate a mass flow. Need to measure the flow of natural gas by using methods that are defined by AGA (or other standard or authority having jurisdiction), to calculate a mass flow. Need to measure the flow of solids or liquids, or of gases that do not behave (approximately) as an Ideal Gas. The P_PTComp instruction uses the Ideal Gas Law ($PV = nRT$) to calculate flow at standard conditions.
Tank Strapping Table (P_StrapTbl)	<p>This instruction calculates the volume of product in an upright cylindrical tank when given the level of the product and the tank calibration table. This instruction can optionally compensate for free water at the bottom of the tank (given a product/water interface level) or for thermal expansion of the tank shell (given the coefficient of linear expansion of the shell material and product and ambient temperatures).</p> <p>The P_StrapTbl instruction also can optionally compensate for a floating tank roof if the product density is provided.</p> <p>This instruction is intended only as a calculation function, between other blocks, and no HMI components are provided.</p>	<ul style="list-style-type: none"> Want to determine the volume of product in an upright cylindrical tank given its level. Have the Strapping Table (tank calibration table) data for the tank. 	<p>This instruction interpolates level between strapping table calibration points linearly. Do not use this instruction if you have a spherical tank, a conical tank, or other tank shape for which the volume does not vary more-or-less linearly with level.</p> <p>To determine the mass of product in the tank, take the volume result from the P_StrapTbl instruction and compensate for product density with further calculations. See the American Petroleum Institute (API) Manual of Petroleum Measurement Standards (MPMS) for the appropriate calculations.</p>

Table 1 - I/O Processing Objects

Process Object	Description	When to Use	When Not to Use
HART Analog Input (P_AInHART)	<p>This instruction monitors one analog input from a flow, level, pressure, temperature, or other HART-connected analog sensor.</p> <p>Alarms are provided when the analog value exceeds user-specified thresholds (high and low).</p> <p>The instruction also provides capabilities for linear scaling of an analog input value from raw (input) units to engineering (output) units. Entry of a substitute PV, which handles an out-of-range or faulted input, is included.</p>	<ul style="list-style-type: none"> • Use an instance of the P_AInHART instruction for each defined HART analog input channel with a connected field device (transmitter). • This instruction provides standard analog input functionality, plus digital HART values, status, enumerations, and diagnostics. • Provides Maintenance selection of the substitute value function to allow manual override of the analog input signal (AV). • Monitors input quality and communication status. Provides value and indication of source and quality for the input signal and the final AV value. 	<ul style="list-style-type: none"> • Do not use with modules other than the supported HART Analog I/O modules included in the library. • For FOUNDATION Fieldbus, use a 1788-EN2FFR module and the P_AInFFR Add-On Instruction • For Profibus, use a 1788-EN2PAR module and the P_AInPAR Add-On Instruction • For simple 4 - 20 milliamp analog inputs, use the P_AIn Add-On Instruction • For simple 4 - 20 milliamp analog outputs, use the P_AOut Add-On Instruction
HART Analog Output (P_AOutHART)	<p>This instruction manipulates an analog output to control a field device, such as a control valve or a motorized gate positioner. The output responds to an Operator (manual) or Program setting of the Controlled Variable (CV) signal.</p> <p>The P_AOutHART instruction controls the analog output in various command sources (Operator, Program, Override, Maintenance, Hand), monitoring for fault conditions.</p>	<p>Use an instance of this instruction for each defined HART analog output channel with a connected field device (actuator).</p> <p>This instruction outputs an analog controlled variable (CV) to the field device. The instruction receives values from the module and the field device.</p> <p>Other than the HART-specific functions, the P_AOutHART instruction functions much like the basic P_AOut instruction.</p>	

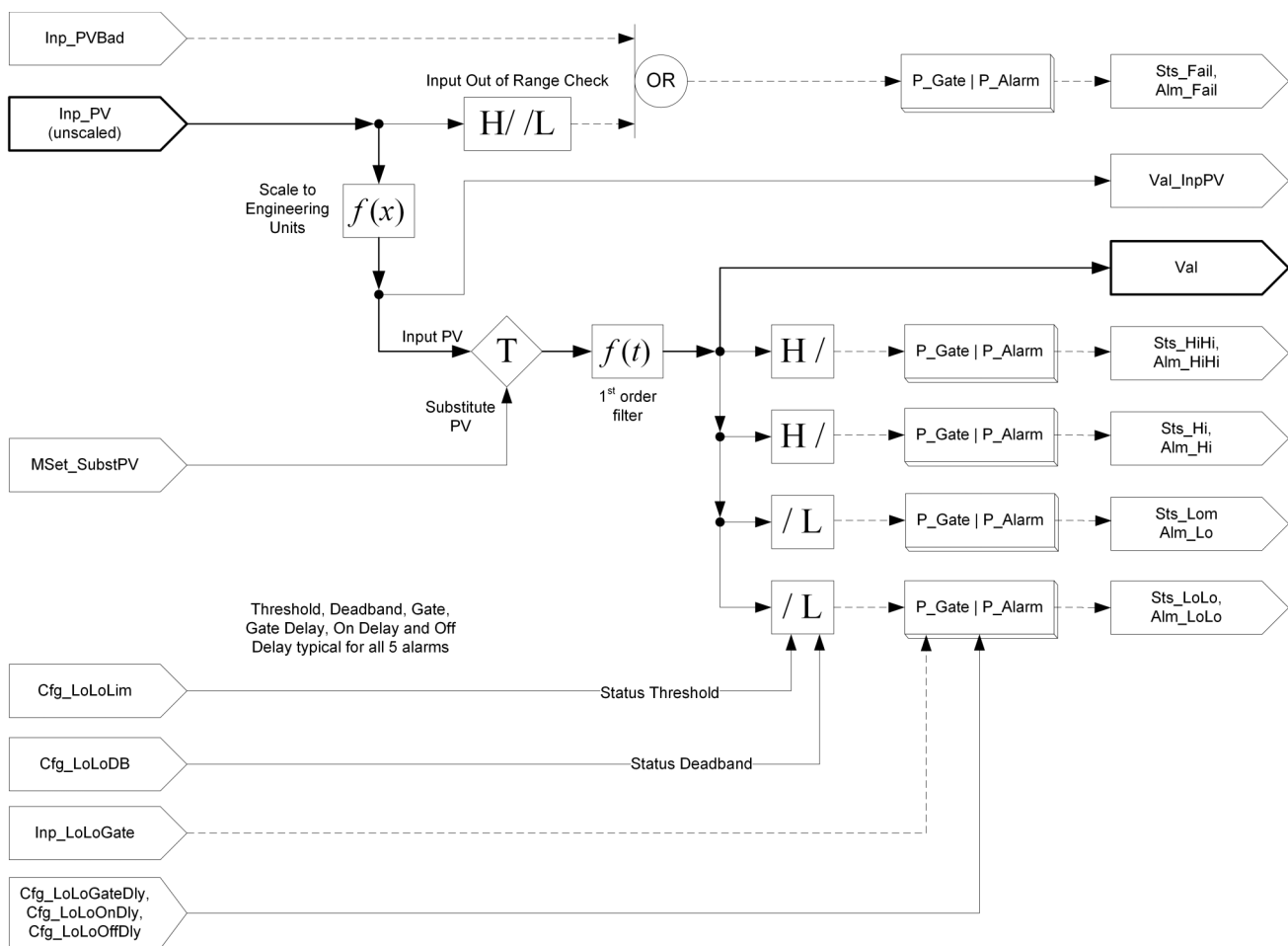
Basic Analog Input (P_AIn)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_AIn (Basic Analog Input) Add-On Instruction monitors one analog value, typically from a channel of an analog input module, and provides alarms when the analog value exceeds user-specified thresholds (high and low). The Analog Input instruction also provides capabilities for linear scaling of an analog input value from raw (input) units to engineering (output) units, and entry of a substitute Process Variable, providing handling of an out-of-range or faulted input.

To keep the instruction memory and execution footprint small, certain capabilities, used less frequently, are reserved for the Advanced Analog Input Add-On Instruction.

The functional diagram for the Basic Analog Add-On Instruction shows the primary functions of scaling to engineering units and providing input alarms.



Functional Description

The Basic Analog Input instruction provides the following capabilities:

- Scales an analog input from raw to engineering units and optionally filters the signal.
- Provides High-High, High, Low and Low-Low status and alarms with configurable delay times and deadbands.
- Provides Program and Operator settings for status thresholds.
- Provides input failure checking for out-of-range High and out-of-range Low, plus PV quality and alarm on failure.
- Provides Maintenance selection of the substitute PV function to allow manual override of the input signal (PV).
- Monitors input quality and communications status. Provides value and indication of source and quality for the input signal and the final PV value.
- Uses a standard command source model (P_CmdSrc instruction) to provide command source (ownership) selection. Required Files

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_AIn_4.10.00_AOIL5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Operations

This section describes the primary operations for this Add-On Instruction.

Alarms

This instruction uses the following alarms, which are implemented by using embedded P_Alarm and P_Gate Add-On Instructions.

Table 2 - P_AIn Alarms

Alarm Name	P_Alarm Name	P_Gate Name	Description
Fail	Fail	FailGate	Raised when any of the following is true: <ul style="list-style-type: none"> • The PV quality is bad • The Inp_PVBad input is true • The PV is outside the configured failure limits • The PV is infinite or not a number (floating point exception) • The raw or engineering unit range configuration is invalid
High PV	Hi	HiGate	Raised when the PV is above the configured High threshold. The threshold deadband, gating, and timing are set in configuration.
High-High PV	HiHi	HiHiGate	Raised when the PV is above the configured High-High threshold. The threshold deadband, gating, and timing are set in configuration.
Low PV	Lo	LoGate	Raised when the PV is below the configured Low threshold. The threshold deadband, gating, and timing are set in configuration.
Low-Low PV	LoLo	LoLoGate	Raised when the PV is below the configured Low-Low threshold. The threshold deadband, gating, and timing are set in configuration.

Execution


The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	The P_AIn Instruction shows a status of bad quality (Sts_PVBad) and an indication on the HMI. All alarms are cleared. Calculation of the scaled Val_InpPV is executed to indicate to the operator the actual input value, even though the primary PV (Val) is not updated (holds last value).
Powerup (prescan, first scan)	Any commands that are received before first scan are discarded. Embedded P_Alarm instructions are handled in accordance with their standard power-up procedures. See the Reference Manual for the P_Alarm Instruction for more information.
Postscan (SFC transition)	No SFC postscan logic is provided.

Simulation

Simulation in P_AIn disables the normal input (Inp_PV) and provides an input on the Operator faceplate for you to enter your own input value.

You must set the Inp_Sim parameter in the controller to '1' to enable simulation.

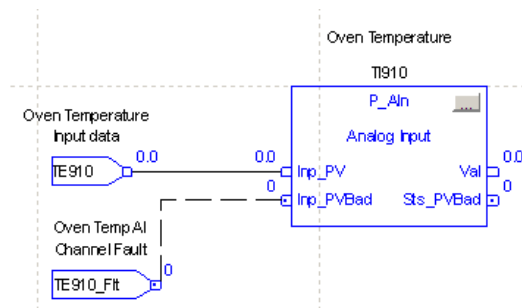
The Simulation icon  is displayed at the top left of the Operator faceplate indicating the device is in simulation.

See the Logix 5000™ Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#), for more information.

Programming Example

This example uses the P_AIn instruction to read a temperature sensor that is also used elsewhere in logic to control the heating element of a chamber.

The Inp_PV parameter must be connected to the value coming from the temperature transmitter. The fault status for the associated I/O channel in the I/O module must be connected to the bad status input, Inp_PVBad.



The output parameters Val and Sts_PVBad can then be connected to the PV and PVFault parameters of a PIDE instruction for control.

To implement this example, the following configuration input parameters need to be set. Those not listed can be left at their default. There is no alarming used in this example.

- Cfg_InpRawMin, Cfg_EUMin: 0 (engineering low range of temperature)
- Cfg_InpRawMax, Cfg_EUMax: 300 (engineering high range of temperature)

In addition, the following strings are configured to drive the display and faceplate:

- Cfg_Desc: Oven Temperature
- Cfg_EU: Deg C
- Cfg_Label: Oven Temp
- Cfg_Tag: TI910

The strings that are listed above are local tags that can be configured through the HMI faceplates or in Logix Designer application by opening the Instruction Logic of the Add-On Instruction instance and then opening the Data Monitor on a local tag.

Analog Input Channel (P_AIChan)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_AIChan Add-On Instruction monitors one analog input channel for the following conditions:

- Invalid configuration
- I/O module fault
- Input out of range
- Instrument reports the following conditions:
 - Out of specification (uncertain)
 - Function check (substitute PV entered manually)
 - Maintenance required
- Channel fault
- Input not-a-number (floating point exception)
- Input stuck (unchanging)

For each condition, the Process Variable (PV) quality to report can be configured as follows:

- Good
- Uncertain
- Bad (raises Fail alarm)

This instruction is usually associated with other instructions, with one instance being used for each analog input of the associated instruction.

This instruction can be integrated with the following instructions in the Rockwell Automation Library of Process Objects:

- Basic Analog Input (P_AIn)
- Advanced Analog Input (P_AInAdv)
- Dual Sensor Analog Input (P_AInDual)
- Multiple Analog Input (P_AInMulti)
- Dosing (P_Dose)

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_AIChan_4.10.00_AOIL5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Operations

This section describes the primary operations for this Add-On Instruction.

Alarms

This instruction uses the following alarms, which are implemented by using embedded P_Alarm and P_Gate Add-On Instructions.

Alarm Name	P_Alarm Name	P_Gate Name	Description
Fail	Fail	FailGate	<p>Raised when any of the following is true:</p> <ul style="list-style-type: none"> The PV quality is bad The Inp_PVBad input is true The PV is outside the configured failure limits The PV is infinite or not a number (floating point exception) The raw or engineering unit range configuration is invalid
		None	<p>Raised when the PV is flagged as Bad, or when the PV is flagged as Uncertain and Cfg_FailOnUncertain is 1.</p> <p>The PV can be configured to be flagged as Bad or Uncertain for the following reasons:</p> <ul style="list-style-type: none"> Its value has not changed for more than the configured Stuck PV time It is outside the configured failure range thresholds for more than the Out of Range on-delay time It is infinite or not a number (floating point exception) Module Fault input is true Channel Fault input is true Out of Specification (measurement uncertain) input is true Function Check (PV substituted at device) input is true Maintenance Required input is true There is a Configuration Error (see Sts_Err and the Err_ bits)

Parameters of the P_Alarm object can be accessed by using the following convention: [P_Alarm Name].[P_Alarm Parameter].

TIP In P_AIChan, the 'Fail' alarm is set not to exist by default. If you set this alarm to exist, be aware the P_AIChan object does not have its own display elements. Thus, you do not have a graphic symbol with flashing border to show and click to call up the P_AIChan faceplate. However, you can still get to the P_AIChan faceplate easily when it raises a failure alarm in two different ways:

- Tie the SrcQ output parameter of the P_AIChan instance to the Inp_PV SrcQ input parameter of the downstream object and enable the I/O failure alarm for that object. Enable navigation from the object to its upstream Channel object. When the P_AIChan raises its Fail alarm, the border for the downstream object blinks. Call up the faceplate for the downstream object, then navigate from there to the P_AIChan faceplate.
- In the FactoryTalk® Alarms and Events alarm setup, create a command string for the P_AIChan 'Fail' alarm that opens the P_AIChan faceplate with that instance's tag. Instructions for how to create the command string are included in publication [PROCES-RM002](#). When the alarm occurs, open the Alarm Summary screen. Double-click the alarm in the summary list and the P_AIChan faceplate is displayed.

Simulation

The Analog Input Channel Add-On Instruction does not have a Simulation capability.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	Clear any received commands. Reset internal timers. Clear the Fail alarm. Flag input quality as 'bad'. Show alarm inhibited status. Other parameters are left in their last state.
Powerup (prescan, first scan)	Reset internal timers. Clear any received commands.
Postscan (SFC Transition)	No SFC postscan logic is provided.

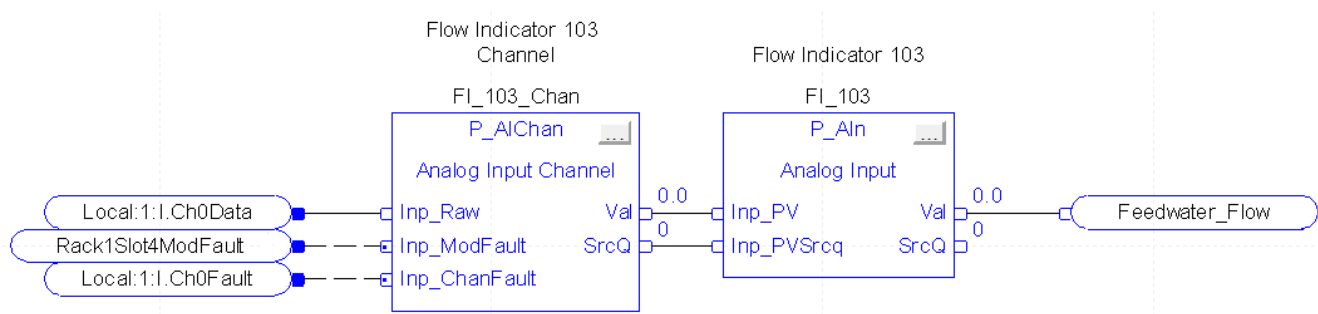
See the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#), for more information.

Programming Example

The following example provides shows the connection from raw analog input through process value by using the P_AIChan block.

The raw input value (Local:1:I.Ch0Data) from the analog input card is used as the raw input value (Inp_Raw) for the P_AIChan block. The output value (Val) and quality (SrcQ) from the P_AIChan block are used as inputs for the P_AIn block. In this configuration, the P_AIn block uses the Cfg_HasChanObj configuration parameter. The final output process value (Feedwater_Flow) is the fully converted, scaled, and filtered analog value that is propagated through the system.

The P_AIChan block also uses the Channel Fault and Module Fault parameters taken from the same analog input module as the process value. Inp_ChannFault is simply the tag value for the channel (Local:1:I.Ch0Fault). The Inp_ModFault parameter (Rack1Slot4ModFault) is generated by using a GSV to the module object (with the instance for the appropriate card and then the EntryStatus parameter). The top four bits of the EntryStatus parameter are checked to make sure that they do not equal 2#0100_xxxx_xxxx_xxxx. The 0100 pattern indicates that the connection is “Running”. All other values are considered faulted.

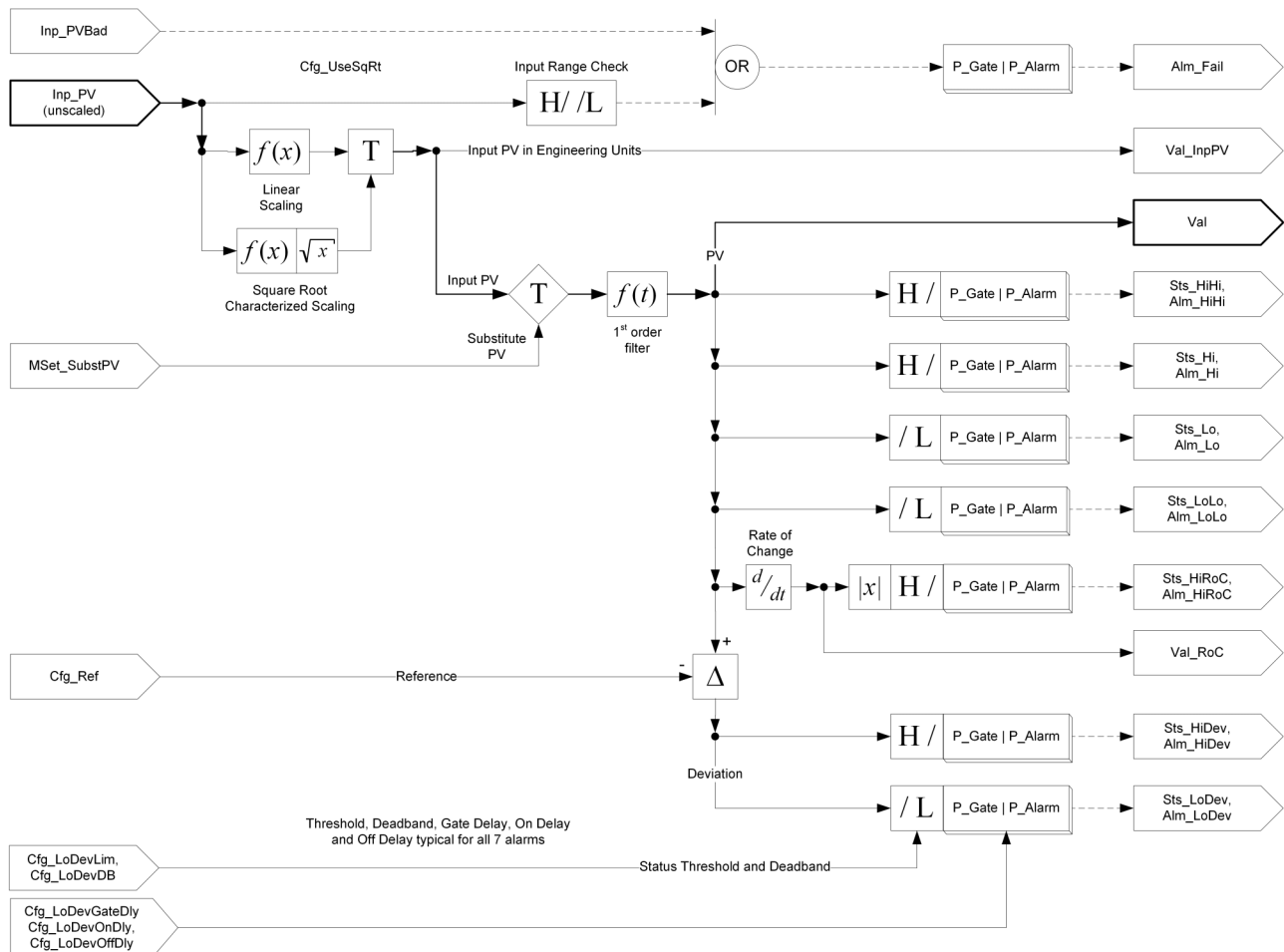


Advanced Analog Input (P_AInAdv)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_AInAdv (Advanced Analog Input) Add-On Instruction monitors one analog value, typically from an Analog Input I/O module.

The diagram shows the functional characteristics of the P_AInAdv Add-On Instruction.



Functional Description

The P_AInAdv instruction provides the following capabilities:

- Linear scaling of the input value from raw (input card) units to engineering (display) units.

- High-High, High, Low, and Low-Low PV alarms with Operator- or Program-entered limits and configurable deadband and delay per alarm.
- Input Source and Quality monitoring of PV uncertain and PV bad inputs, plus monitoring of the PV for out-of-range condition and alarming on PV failure.
- PV filtering (first-order) to reduce signal noise.
- Maintenance capability to enter a substitute PV.
- Graphic Symbols, plus a faceplate with bar graph PV indication, command source selection, alarm limit entry and alarm display, configuration and acknowledgement, trending, and Maintenance and Engineering configuration and setup.
- Advanced features (unique to the P_AInAdv Advanced Analog Input instruction):
 - Square root characterized scaling of the input value from raw (input card) units to engineering (display) units. Square root characterized scaling is used with orifice plates or other pressure-differential elements for flow measurement when the transmitter does not provide square root characterization. The square root scaling in the P_AInAdv instruction works with \pm pressure differential to provide positive or negative flow values.
 - Configurable reference (setpoint) value with configurable alarms for PV deviation above or below the reference value.
 - Calculation of the PV rate of change and configurable high rate of change alarming.

Each of the advanced features can be individually selected on or off for a given PV (instruction instance).

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_AInAdv_4.10.00_AOIL5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Operations

This section describes the primary operations for this Add-On Instruction.

Alarms

This instruction uses the following alarms, which are implemented by using embedded P_Alarm and P_Gate Add-On Instructions.

Table 3 - P_AInAdv Alarms


Alarm Name	P_Alarm Name	P_Gate Name	Description
Fail	Fail	FailGate	Raised when the PV is outside the configured failure limits.
		None	Raised when any of the following is true: <ul style="list-style-type: none"> • The PV quality is bad • The Inp_PVBad input is true • The PV is infinite or not a number (floating point exception) • The raw or engineering unit range configuration is invalid
High Deviation	HiDev	HiDevGate	Raised when the amount by which the PV exceeds the setpoint or reference is above the High Deviation threshold. The threshold deadband, gating, and timing are set in configuration.
High PV	Hi	HiGate	Raised when the PV is above the High threshold. The threshold deadband, gating, and timing are set in configuration.
High Rate of Change	HiRoC	HiRoCGate	Absolute value of PV rate of change exceeds High Rate of Change limit. Limit set by Operator. Deadband and severity are set in configuration.
High-High PV	HiHi	HiHiGate	Raised when the PV is above the High-High threshold. The threshold deadband, gating, and timing are set in configuration.
High Rate of Change	HiRoC	HiRoCGate	Raised when the rate at which the PV is increasing or decreasing exceeds the High-Rate of Change threshold. The threshold deadband, gating, and timing are set in configuration.
Low PV	Lo	LoGate	Raised when the PV is below the Low threshold. The threshold deadband, gating, and timing are set in configuration.
Low Deviation	LoDev	LoDevGate	Raised when the amount by which the PV exceeds the setpoint or reference is below the Low Deviation threshold. (Since the threshold is a negative number, this value is the amount the PV falls below the setpoint or reference.) The threshold deadband, gating, and timing are set in configuration.
Low-Low PV	LoLo	LoLoGate	Raised when the PV is below the Low-Low threshold. The threshold deadband, gating, and timing are set in configuration.

Parameters of the P_Alarm object can be accessed by using the following convention: [P_Alarm Name].[P_Alarm Parameter].

Simulation

Simulation in P_AInAdv disables the normal input (Inp_PV) and provides an input on the Operator faceplate for you to enter your own input value.

You must set the Inp_Sim parameter in the controller to '1' to enable simulation.

The Simulation icon  is displayed at the top left of the Operator faceplate indicating the device is in simulation.

When you have finished in simulation, set the Inp_Sim parameter in the controller to '0' to return to normal operation.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	The P_AInAdv instruction shows a status of bad quality (Sts_PVBad) on the HMI. All alarms are cleared. Calculation of the scaled input PV value is executed to indicate to the operator the actual input value, even though the primary PV (Val) is not updated (holds last value).
Powerup (prescan, first scan)	Any commands that are received before first scan are discarded. Embedded P_Alarm instructions are handled in accordance with their standard power-up procedure. See the reference manuals for the P_Alarm instructions for more information.
Postscan (SFC transition)	No SFC postscan logic is provided.

See the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#), for more information.

Programming Example

For a generic programming example, see the [Basic Analog Input \(P_AIn\)](#) section.

The P_AInAdv Add-On Instruction has the following advanced features that are not included in the generic programming example.

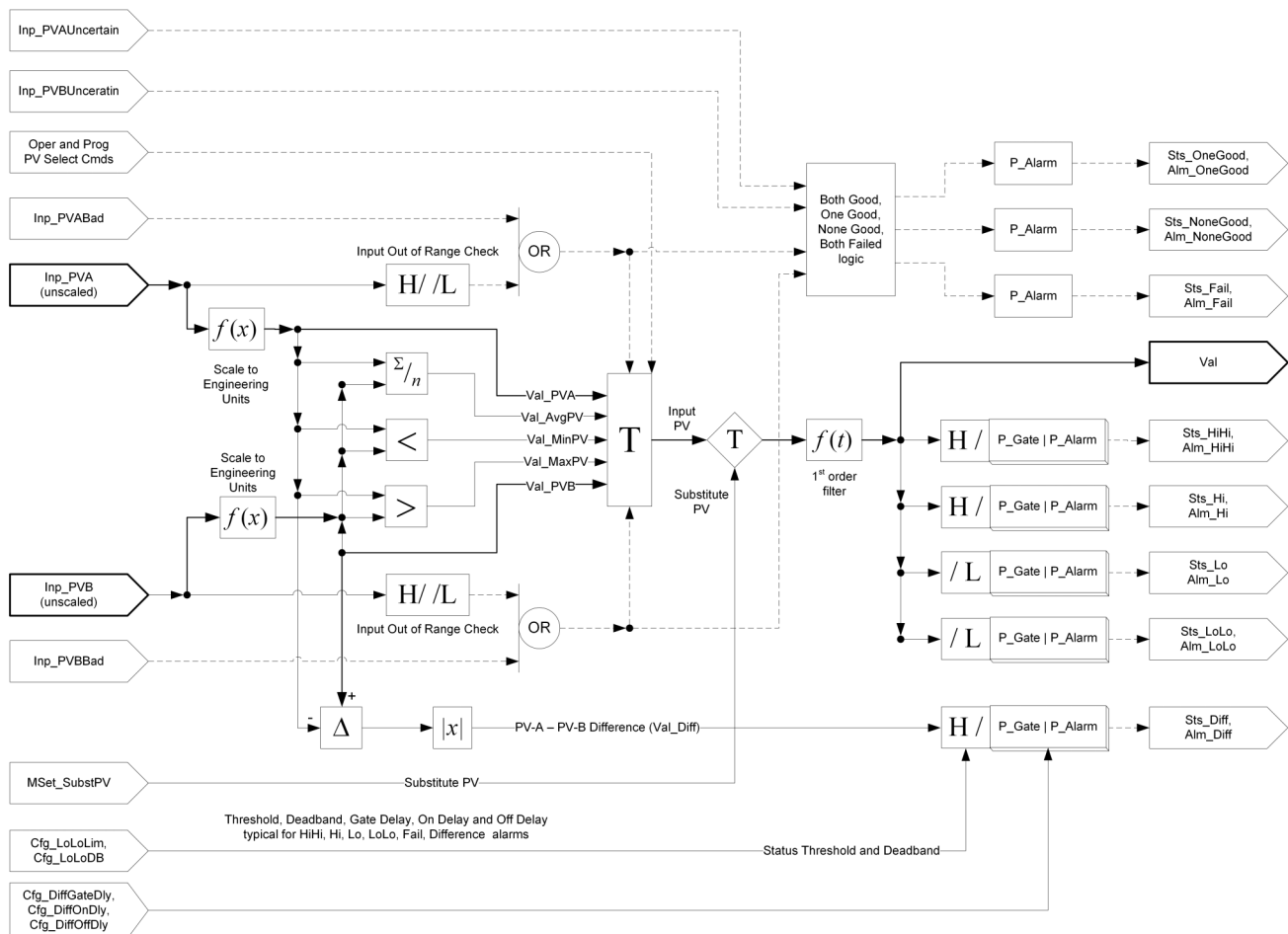
- Square root characterization
- Deviation display/alarms
- Rate of change display/alarms

Dual Sensor Analog Input (P_AInDual)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_AInDual (Dual Analog Input) Add-On Instruction monitors one analog Process Variable (PV) by using two analog input signals (dual sensors, dual transmitters, and dual-input channels).

The diagram shows the functional characteristics of the P_AInDual Add-On Instruction.



Functional Description

The P_AInDual instruction provides the following capabilities:

- Linear scaling of each input signal from raw (input card) units to engineering (display) units.

- Operator or Program selection of the sensor/input A value, the sensor/input B value, the average of the two, the lesser of the two, or the greater of the two as the PV value.
- High-High, High, Low, and Low-Low PV alarms with configurable threshold, deadband, and delay per alarm.
- Input Source and Quality monitoring for uncertain or bad input for each sensor/transmitter/input. Monitoring of each signal for out-of-range condition (if one PV is bad, failed, or out of range, the other PV is automatically selected).
- Warning alarm if the difference between the PV of the two sensors exceeds a configured limit.
- Warning alarm if only one PV has good quality.
- Warning alarm if neither PV has good quality (for example, if both are uncertain).
- Failure alarm if both PVs are bad—for example, each PV has bad quality (Inp_PVABad/Inp_PVBBad) or is outside of the configured failure range.
- Filtering (first order) of the selected PV to reduce process or electrical signal noise.
- Maintenance capability to enter a substitute PV.
- Graphic symbols, plus a faceplate with bar graph PV indication, command source selection, alarm limit entry and alarm display, configuration and acknowledgement, trending, and Maintenance and Engineering configuration and setup.

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_AInDual_4.10.00_AOIL5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Operations

This section describes the primary operations for this Add-On Instruction.

Alarms

This instruction uses the following alarms, which are implemented by using embedded P_Alarm and P_Gate Add-On Instructions.


Table 4 - P_AInDual Alarms

Alarm Name	P_Alarm Name	P_Gate Name	Description
Difference	Diff	DiffGate	Raised when the difference between the two input signals exceed the configured high difference threshold.
Fail	Fail	FailGate	Raised when any of the following is true: <ul style="list-style-type: none"> Both PV input values are outside the configured failure range thresholds Both PV input values have bad quality or are infinite or not a number Selected PV is infinite or not a number (floating point exception) Raw or engineering unit range configuration is invalid
High PV	Hi	HiGate	Raised when the PV is above the High threshold. The threshold deadband, gating, and timing are set in configuration.
High-High PV	HiHi	HiHiGate	Raised when the PV is above the High-High threshold. The threshold deadband, gating, and timing are set in configuration.
Low PV	Lo	LoGate	Raised when the PV is below the Low threshold. The threshold deadband, gating, and timing are set in configuration.
Low-Low PV	LoLo	LoLoGate	Raised when the PV is below the Low-Low threshold. The threshold deadband, gating, and timing are set in configuration.
NoneGoodPV	NoneGood	None	Raised when neither PV input has good quality. This alarm is an indication that both PV inputs have degraded or bad signal quality, and so the resulting PV does not have good quality.
Only One Good PV	OneGood	None	Raised when either of the two PV inputs has degraded or bad quality.

Parameters of the P_Alarm object can be accessed by using the following convention: [P_Alarm Name].[P_Alarm Parameter].

Simulation

Simulation in P_AInAdv disables the normal input (Inp_PVA and Inp_PVB) and provides an input on the Operator faceplate for you to enter your own A and B input values.

You must set the Inp_Sim parameter in the controller to '1' to enable simulation. The Simulation icon  is displayed at the top left of the Operator faceplate indicating the device is in simulation.

When you have finished in simulation, set the Inp_Sim parameter in the controller to '0' to return to normal operation.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	The P_AInDual instruction shows a status of bad quality (Sts_PVBad) on the HMI. All alarms are cleared. Calculation of the scaled input PV value is executed to indicate to the operator the actual input value, even though the primary PV (Val) is not updated (holds last value).
Powerup (prescan, first scan)	Any commands that are received before first scan are discarded. Embedded P_Alarm instructions are handled in accordance with their standard power-up procedure. See the reference manuals for the P_Alarm instructions for more information.
Postscan (SFC transition)	No SFC postscan logic is provided.

See the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#), for more information.

Programming Example

For a generic programming example, see the [Basic Analog Input \(P_AIn\)](#) section.

The P_AInDual Add-On Instruction has the following advanced features that are not included in the generic programming example.

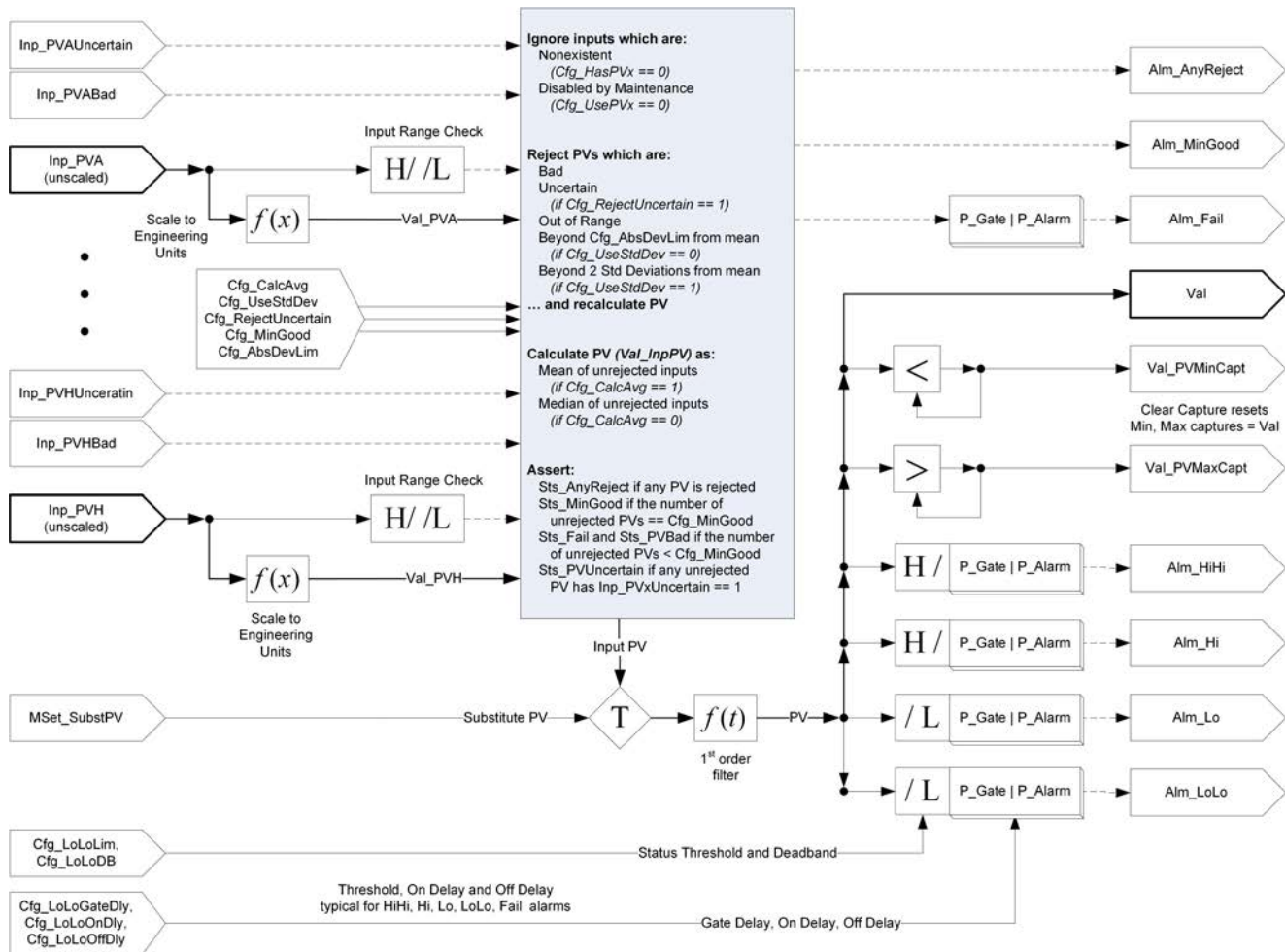
- Dual-inputs
- Alarm if difference between the two input PVs exceeds a configured limit

Multiple Analog Input (P_AInMulti)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_AInMulti (Multiple Analog Input) Add-On Instruction monitors one analog process variable (PV) by using up to eight analog input signals (sensors, transmitters, input channels).

The diagram shows the functional characteristics of the P_AInMulti Add-On Instruction.



Functional Description

The Multiple Analog Input Instruction provides the following capabilities:

- Configuration to use 2...8 input signals.
- Linear scaling of each input signal from raw (input card) units to engineering (display) units.

- Input Source and Quality monitoring of inputs, plus monitoring of each signal for out-of-range condition. Rejection from the PV calculation of inputs that are out of range, flagged as bad, infinite, or not a number (floating point exception values).
- First-order filtering of the calculated PV to reduce process or electrical signal noise.
- Calculation of the average (mean) or median of the inputs in use as the PV value.
- Selectable rejection from the PV calculation of inputs that are outside of two standard deviations from the mean (minimum four inputs required), or inputs that are outside of a user-defined deviation from the mean.
- Configuration of the minimum number of good (unrejected) input signals required to have a good PV value, and an alarm if the required number of good inputs is not met.
- Configuration of which PV to use if there are only two unrejected signals remaining: the lesser, the greater, or the average of the two.
- Maintenance capability to enter a substitute PV.
- High-High, High, Low, and Low-Low PV threshold alarms for the overall calculated PV, with configurable threshold, deadband, on-delay, and off-delay per alarm.
- An alarm if any inputs configured to be used are rejected.
- An alarm if the number of unrejected inputs is equal to the minimum number required to be good (meaning the next input failure results in a PV failure).
- Display elements, plus a faceplate with bar graph PV indication, command source selection, alarm limit entry and alarm display, configuration, acknowledgement, trending, and maintenance and engineering configuration and setup.

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The **P_AInMulti_4.10.00_AOIL5X** Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Operations

This section describes the primary operations for this Add-On Instruction.

Alarms

This instruction uses the following alarms, which are implemented by using embedded P_Alarm and P_Gate Add-On Instructions.


Alarm Name	P_Alarm Name	P_Gate Name	Description
Any Reject	AnyReject	None	<p>Raised when at least one input signal has been rejected because of any of the following:</p> <ul style="list-style-type: none"> • It is outside the configured failure range. • It is a statistical outlier per the Modified Thompson Tau test. • It is outside of a user-defined deviation from the mean. • It has its Bad quality input bit set or its Source and Quality input indicates it has Bad quality. • It has a floating point value that is infinite or not a number (floating point exception).
Fail	Fail	FailGate	<p>Raised when any of the following is true:</p> <ul style="list-style-type: none"> • Number of unrejected PVs is less than the configured required number of good PVs • Calculated PV is infinite or not a number (floating-point exception) • Raw or engineering unit range configuration is invalid <p>A PV can be rejected if:</p> <ul style="list-style-type: none"> • It is set to not be used (by Maintenance) • It is outside the configured failure range thresholds • It is infinite or not a number (floating point exception) • It has Bad quality • It has Uncertain quality and Cfg_RejectUncertain is true • It is an outlier, either because its deviation is outside the configured threshold from the mean or its deviation from the mean exceeds the Modified Thompson Tau statistical test
High PV	Hi	HiGate	<p>Raised when the PV is above the High threshold. The threshold deadband, gating, and timing are set in configuration.</p>
High-High PV	HiHi	HiHiGate	<p>Raised when the PV is above the High-High threshold. The threshold deadband, gating, and timing are set in configuration.</p>

Alarm Name	P_Alarm Name	P_Gate Name	Description
Low PV	Lo	LoGate	Raised when the PV is below the Low threshold. The threshold deadband, gating, and timing are set in configuration.
Low-Low PV	LoLo	LoLoGate	Raised when the PV is below the Low-Low threshold. The threshold deadband, gating, and timing are set in configuration.
Minimum Good	MinGood	None	Raised when at least one input signal has been rejected, and the remaining unrejected signals are the minimum number configured as required for a good PV. This status/alarm is to warn you that the next input failure results in a Bad PV quality.

Parameters of the P_Alarm object can be accessed by using the following convention: [P_Alarm Name].[P_Alarm Parameter].

Simulation

Simulation in P_AInMulti disables the normal inputs (Inp_PVA through Inp_PVH) and provides up to eight inputs on the Operator faceplate for you to enter your own input values.

You must set the Inp_Sim parameter in the controller to '1' to enable simulation. The Simulation icon  is displayed at the top left of the Operator faceplate indicating the device is in simulation.

When you have finished in simulation, set the Inp_Sim parameter in the controller to '0' to return to normal operation.

Execution

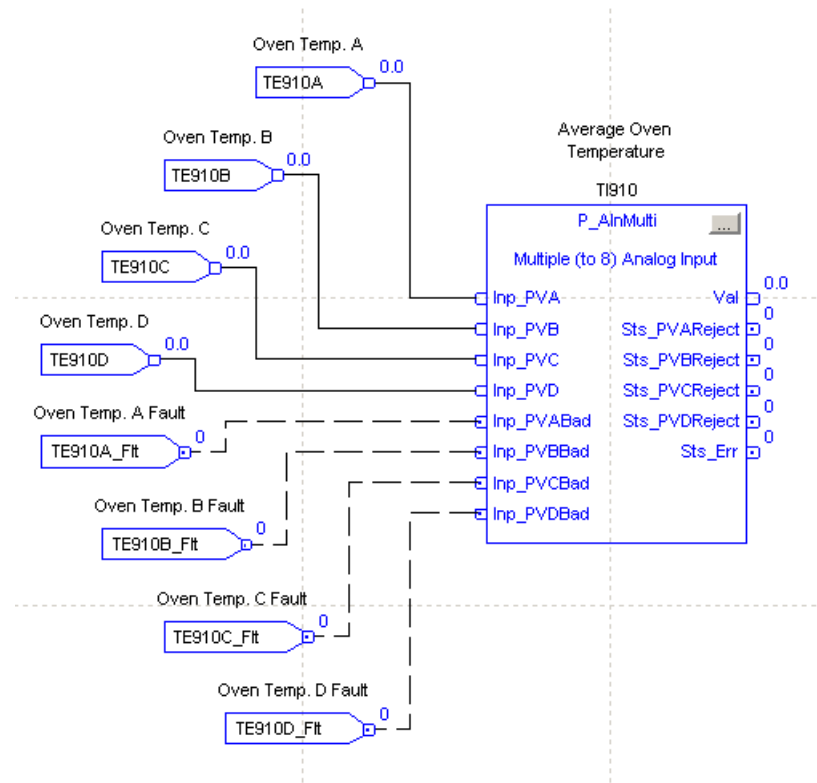
The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	The P_AInMulti instruction shows a status of Bad Quality (Sts_PVBad) on the HMI. All alarms are cleared. Calculation of the scaled input PV value is executed to indicate to the operator the actual input value, even though the primary PV (value) is not updated (holds last value).
Powerup (prescan, first scan)	Any commands received before first scan are discarded. Embedded P_Alarm instructions are handled in accordance with their standard power-up procedure. See the reference manuals for the P_Alarm instructions for more information.
Postscan	No SFC postscan logic is provided.

Programming Example

This example uses the P_AInMulti instruction to average multiple sensors for a single PV.

In this example, there is an oven with four temperature sensors (A, B, C, D). The average of these temperature sensors is used elsewhere in logic to control the heating element of the chamber.



The Inp_PVA, Inp_PVB, Inp_PVC, and Inp_PVD parameters (in the Function Block example above) are connected to the values that is coming from the four temperature transmitters. The fault status of each of these sensors is tied to the bad status input of P_AInMulti (for example, Inp_PVABad).

The output parameters Val and Sts_PVBad could then be connected to the PV and PVFault parameters of a PIDE instruction for control.

To implement this example, the following configuration input parameters need to be set. Those not listed could be left at their default. There is no alarming set in this example.

```

Cfg_HasPVA, Cfg_HasPVB, Cfg_HasPVC, Cfg_HasPVD: 1
Cfg_HasPVE, Cfg_HasPVE, Cfg_HasPVG, Cfg_HasPVH: 0
Cfg_UsePVA, Cfg_UsePVB, Cfg_UsePVC, Cfg_UsePVD: 1
Cfg_UsePVE, Cfg_UsePVE, Cfg_UsePVG, Cfg_UsePVH: 0
Cfg_InpRawMin, Cfg_EUMin: 0 (engineering low range of temperature)
Cfg_InpRawMax, Cfg_EUMax: 300 (engineering high range of temperature)
Cfg_CalcAvg: 1

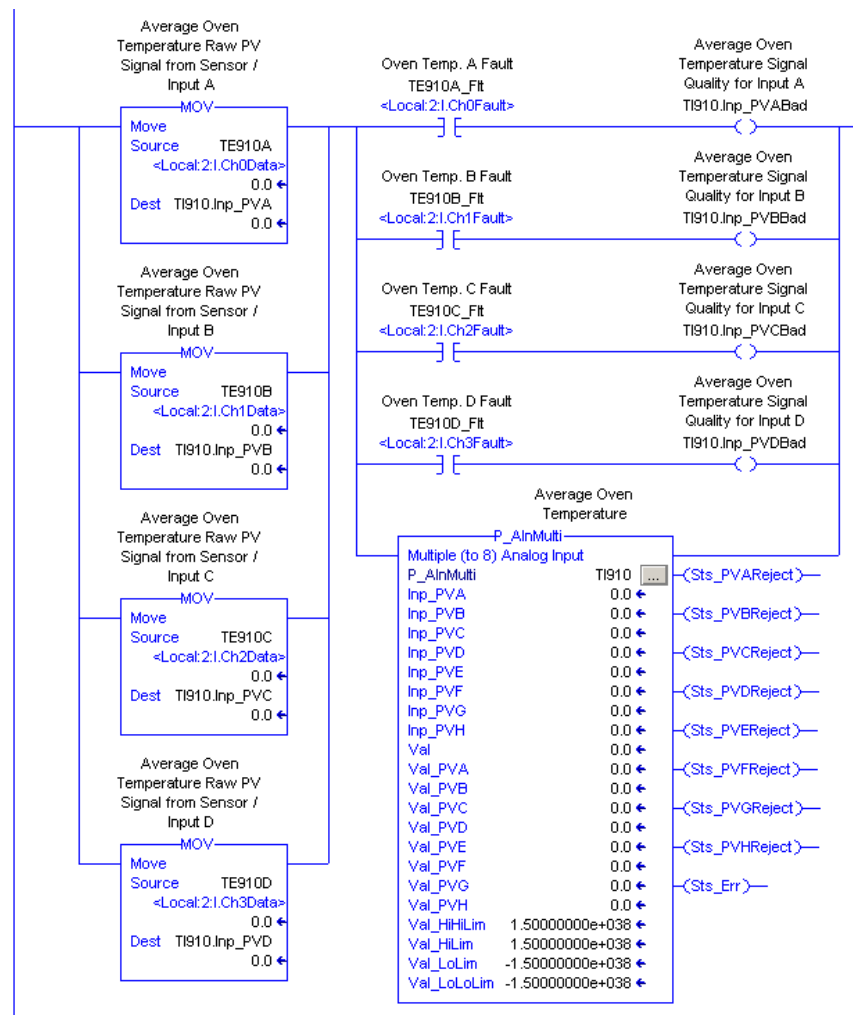
```

If we set the range to 0...300, then Cfg_FailHiLim needs to approximately 310. In addition, the following strings are configured to drive the operation faceplate:

Cfg_Desc: Average Oven Temperature
 Cfg_EU: Deg C
 Cfg_Label: Avg Oven Temp
 Cfg_PVATag: TE910A
 Cfg_PVBTag: TE910B
 Cfg_PVCTag: TE910C
 Cfg_PVDTag: TE910D
 Cfg_Tag: TI910

The above strings are local tags that can be configured through the HMI faceplates or in Logix Designer application by opening the Instruction Logic of the Add-On Instruction instance and then opening the Data Monitor on a local tag.

This Ladder Logic diagram shows the P_AlnMulti instruction in the same example with multiple temperature sensors (inputs A, B, C, D).

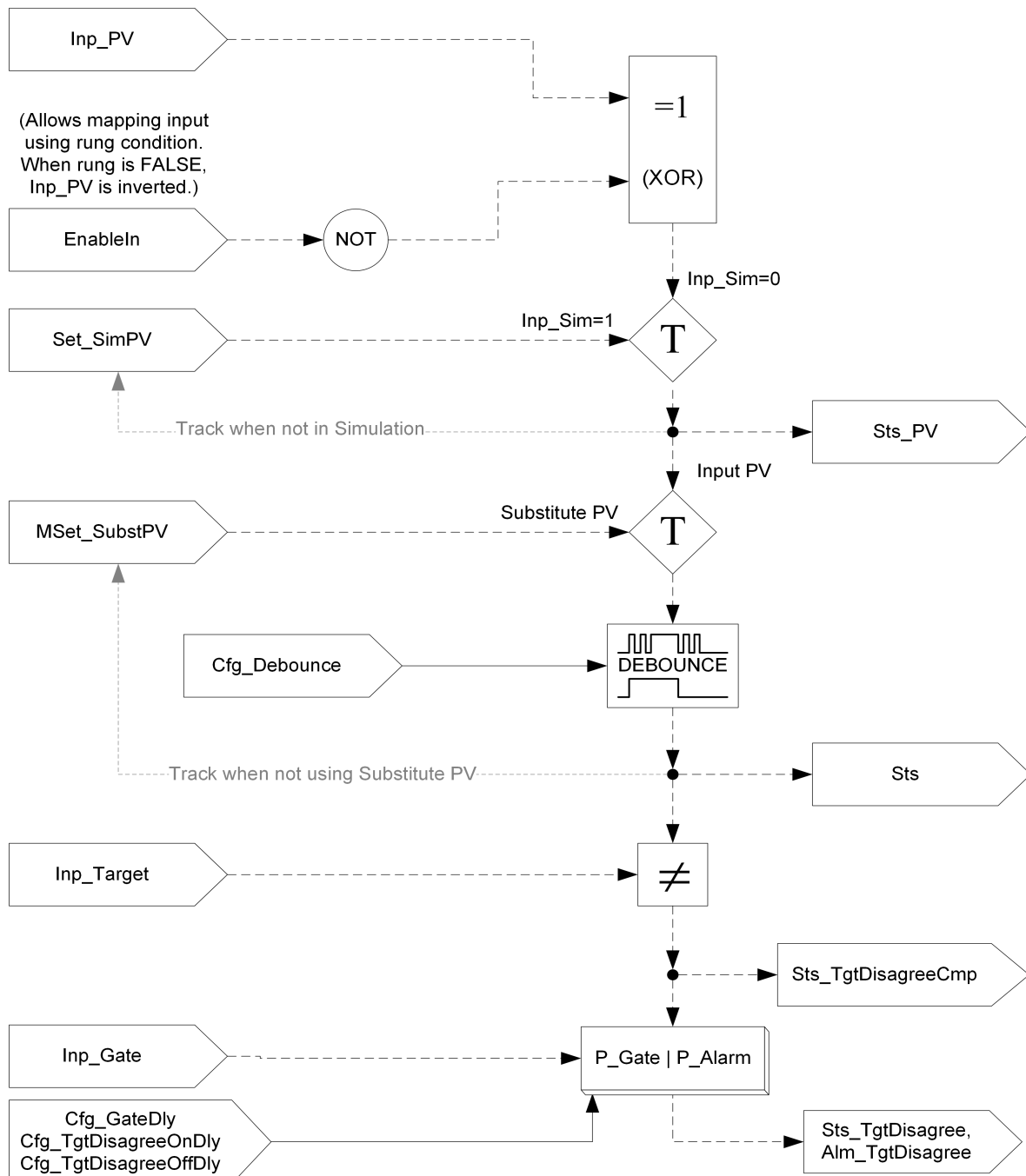


Discrete Input Object (P_DIn)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_DIn (Discrete Input) Add-On Instruction is used to receive and process a single discrete condition (the Process Variable or PV), typically for a channel of a discrete input card. It can be used with any discrete (BOOL) signal.

The diagram shows the functional characteristics of the P_DIn Add-On Instruction.



Functional Description

The P_DIn instruction provides the following capabilities:

- Display of the input state; the 0-state and 1-state names are configurable. The input state is also displayed independently, even when the input is substituted.

- Target Disagree status and optional alarm based on comparing the input state against a target (good) state. The Target Disagree status is enabled by a gating input signal with a configurable gate delay. The Target Disagree status and Alarm On-Delay and Off-Delay are configurable.
- Handle an I/O fault input by displaying the communication fault to the operator and raising an I/O fault alarm.
- Selection and entry of a manual (substitute) PV. This manual override is made clearly visible to the operator.
- Support for a simulated PV for use in instruction testing, demonstration, or operator training.

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The **P_DIn_4.10.00_AOIL5X** Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Operations

This section describes the primary operations for this Add-On Instruction.

Alarms


This instruction uses the following alarms, which are implemented by using embedded P_Alarm and P_Gate Add-On Instructions.

Alarm Name	P_Alarm Name	P_Gate Name	Description
Target Disagree	TgtDisagree	TgtDisagreeGate	Raised when the Input PV is not in the same state as the Target (Inp_Target). Gating, and timing are set in configuration.
I/O Fault	IOFault	-	Raised when the I/O Fault input (Inp_IOFault) is true.

Simulation

Simulation in P_DIn provides a simulated 0-state or 1-state input (Inp_PV) that you can process as if it were an actual input.

You must set the Inp_Sim parameter in the controller to '1' to enable simulation.

The Simulation icon  is displayed at the top left of the Operator faceplate indicating the device is in simulation.

When you have finished in simulation, set the Inp_Sim parameter in the controller to '0' to return to normal operation.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	Processing for EnableIn False (false rung) is handled the same as the main logic routine except that the state of Inp_PV is inverted. This process allows the P_DIn Add-On Instruction in a ladder diagram instance to have its input mapped by using an XIC of the input on the rung with the P_DIn instruction instead of using a separate branch or rung to map the input. Inp_PV is set to 1 (or 0 as appropriate) when using the on-rung mapping.
Powerup (prescan, first scan)	The P_DIn Add-On Instruction uses standard TON timers for status On-delay, Off-delay, and Gate Timing, which reset on Powerup or prescan. As a result, the status initiates as if the Gate input had been changed from 0 to 1.
Postscan (SFC transition)	No SFC postscan logic is provided.

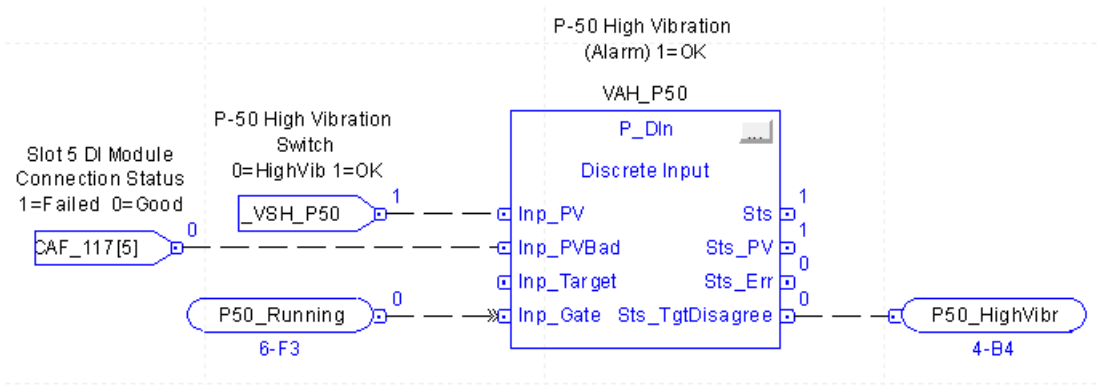
See the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#), for more information.

Programming Example

A simple example of P_DIn use in a function block routine is provided below. In this example, tag I_VSH_P50 is the digital process value being monitored by the P_DIn block. This tag provides a Boolean indication of High Vibration. The bad quality indication for the process value (Inp_PVBad) comes from the connection status indication on the input module.

Inp_Target is defaulted to 1 indicating that the normal condition for I_VSH_P50 is also 1 (tag comments confirm 1=OK for this process value). Inp_Gate is connected to the Motor Running status tag (P50_Running) that comes from the Sts_Running output of the P_Motor block instance for this motor (P50_Motor). When P50_Running is True, On Delay/Off Delay timing is applied to the target disagree status (Sts_TgtDisagree) output of the block (in other words, the output Sts_TgtDisagree will not indicate disagreement/agreement between the Inp_PV target values until after the gate delay/delay times have expired). The gate delay is configured to give the motor sufficient time after starting to settle into full normal speed run before enabling the high vibration indication and alarm. The On Delay/Off Delay values are set as normal configuration parameters.

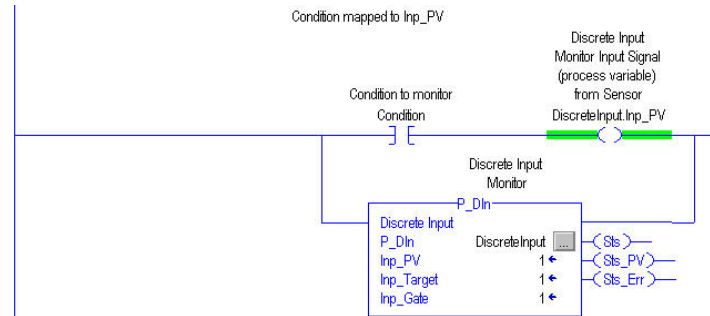
Finally, P50_HighVibr is the output tag that will indicate the status of I_VSH_P50 with appropriate gate delays based on whether the motor is running.



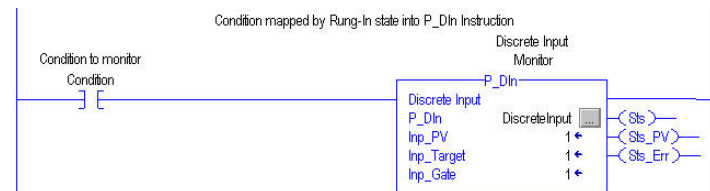
Implementation by Using the EnableIn False Feature

The P_DIn instruction can be used in a ladder diagram routine with the input condition carried by the Rung-In condition instead of being mapped on a separate branch.

The following illustration shows normal implementation with the input condition mapped to Inp_PV on a separate branch.



The following illustration shows EnableIn FALSE implementation with the input condition mapped to the P_DIn instruction by using the Rung-In state.



The Rung-In condition determines whether the normal code ('Logic' Routine) for the Add-On Instruction is executed or its EnableIn False code ('EnableInFalse' Routine) is executed.

In the P_DIn instruction, the EnableIn False code is identical to the Logic code, except it uses the inverse of the Inp_PV signal for processing. To use the Rung-In mapping, method, set Inp_PV to 1 (its default value). When the rung is TRUE, Inp_PV (= 1) is treated as TRUE (not inverted), and when the rung is FALSE, Inp_PV (= 1) is treated as FALSE (inverted).

Discrete Input Object Advanced (P_DInAdv)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

This instruction provides functionality in addition to the content included in the P_DIn Add-On Instruction.

Functional Description

The P_DInAdv instruction provides the following additional capabilities compared to the P_DIn:

- Speed Switch functionality. Allows the use of normal discrete IO (found in legacy plants) to be used as a Speed Switch on equipment powered by e.g. electrical motors. The Speed Switch detects the equipment to be at speed and in normal operation. If the driven element coupling should fail, the Speed Switch indicates the equipment powered to be in an abnormal state while contactor feedback may still indicate the motor in a running state.
- Speed Switch state indication derived from the input state - Accelerating, Run, Decelerating, and Stop.
- Warning and Fail sequential indication with or without alarm indication.
- Drive speed feedback from VSD is used to adjust the base delay times for the Warning and Fail indications.
- Stale input detection - During a stuck input from the field (either high or low) the stale input condition will be detected after 20 seconds and the TargetDisagree state will be triggered. This timer is not user adjustable from the faceplate.
- Run feedback can update the status of the Speed Switch to the Run state before the default Accelerating state has timed out.
- The Run state can be maintained when the Speed Switch is in the Decelerating state.
- Based on the pulsed input, the field equipment speed can be determined.

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_DInAdv_4.10.00_AOI.L5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Operations

This section describes the primary operations for this Add-On Instruction.

Speed Switch

The speed switch setting allows the use of static or dynamic (pulsed) IO to achieve the Speed Switch functionality. For static IO, a selector button allows the use of a transition from 0 to 1 or a transition from 1 to 0 to be used for detection of the "at speed" state of the motor.

A dynamic input may be selected as the signal type to be used for detecting the "at speed" state of the motor. The duty cycle of the pulses does not have to be equal, satisfactory results have been obtained with a duty cycle of 10 / 90 % for a time period of 200ms on a L7 or later CPU.

Warning and Failure Indication

P_DInAdv has two TargetDisagree indications which may be used for a Warning and Failure indication. The Warning is triggered first and is always followed by the Fail indication. Both the Warning and Fail indications are independently configured with delay timers and also configured with or without alarming according to individual alarm severity. This feature is useful in cases where the operator is granted a period to react following a warning before an outright failure is reported.

Speed Switch State Indication

P_DInAdv can display the state of the equipment being monitored. Time in seconds for the various states are configured on the faceplate. The state is determined by monitoring the IO transitions. It is important to note that the state will be reported for the entire duration of the set time. State of "Accelerating" and "Decelerating" will be active for the selected time. During normal operation the state will change from Stop to "Accelerating" to "Run" to "Decelerating" and back to "Stop". The set times are only applicable to "Accelerating" and "Decelerating".

Speed Feedback

If the Warning & Fail indication is selected with the Speed Switch functionality, and the equipment is observed for an at speed condition which is powered by a VSD, the delay times may be adjusted according the VSD speed feedback. A slower speed will have a longer delay before the Warning and Failure is indicated. A higher speed will have delay times closer to the base times entered for the Warning and Fail indication. To adjust the threshold between the low speed and higher speed, the drive low speed threshold may be used as a percentage of maximum speed out of a 100%.

Alarms


This instruction uses the following alarms, which are implemented by using embedded P_Alarm and P_Gate Add-On Instructions.

Alarm Name	P_Alarm Name	P_Gate Name	Description
Target Disagree	TgtDisagreeW	TgtDisagreeWarnGate	Raised when any of the following is true: <ul style="list-style-type: none">The Input PV is not in the same state as the Target (Inp_Target)
Target Disagree	TgtDisagree	TgtDisagreeGate	Raised when the Input PV is not in the same state as the Target (Inp_Target). Gating, and timing are set in configuration.
I/O Fault	IOFault	-	Raised when the I/O Fault input (Inp_IOFault) is true.

Simulation

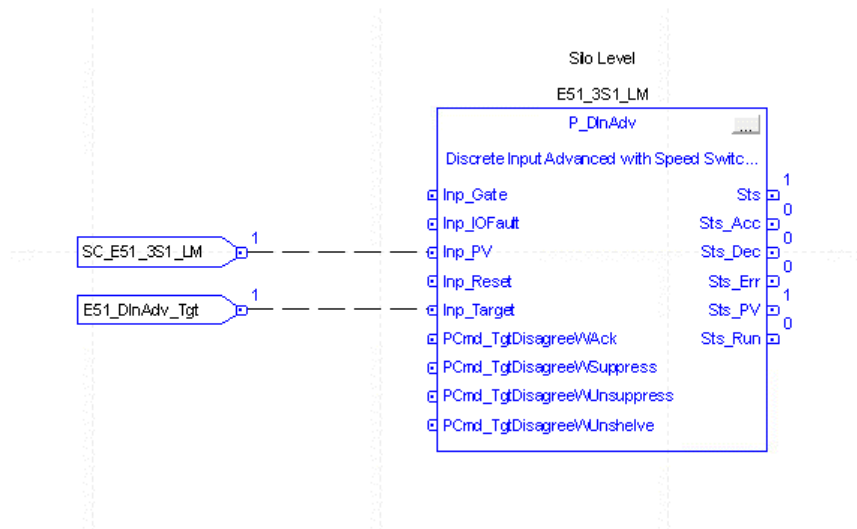
Simulation in P_DInAdv provides a simulated 0-state or 1-state input (Inp_PV) that you can process as if it were an actual input.

You must set the Inp_Sim parameter in the controller to '1' to enable simulation.

The Simulation icon  is displayed at the top left of the Operator faceplate indicating the device is in simulation.

When you have finished in simulation, set the Inp_Sim parameter in the controller to '0' to return to normal operation.

Programming Example



Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	Processing for EnableIn False (false rung) is handled the same as the main logic routine except that the state of Inp_PV is inverted. This process allows the P_DIn Add-On Instruction in a ladder diagram instance to have its input mapped by using an XIC of the input on the rung with the P_DIn instruction instead of using a separate branch or rung to map the input. Inp_PV is set to 1 (or 0 as appropriate) when using the on-rung mapping.
Powerup (prescan, first scan)	The P_DIn Add-On Instruction uses standard TON timers for status On-delay, Off-delay, and Gate Timing, on Powerup or prescan. As a result, the status initiates as if the Gate input had been changed from 0 to 1.
Postscan (SFC transition)	No SFC postscan logic is provided.

See the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#), for more information.

Discrete Output (P_DOut)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The Discrete Output (P_DOut) Add-On Instruction controls a device by a single discrete output signal and optionally monitors feedback from the device to check for device failures. The P_DOut instruction operates in a variety of modes, and can provide steady, single pulsed, or continually pulsed output.

Functional Description

The P_DOut instruction provides the following capabilities:

- Controls one discrete output, with configurable text labels for the On and Off states of the output.
- Provides for the following Operator and Program commands: set the output state to On or Off, pulse the output On once, pulse the output Off once, or set the output to a continuous pulsing operation. Pulse times (on-time and off-time) are configurable.
- Monitors two discrete feedback inputs, monitoring the actual position of the device.
- Detects failure to reach the target state, after a configurable time, and alarms the failure when the feedback inputs are used. Optionally 'sheds' to the de-energized state on a feedback failure.
- Monitors Permissive conditions that enable commanding the device to the On state.
- Monitors Interlock conditions that return the device to its de-energized state (Off).
- Provides simulation of a normally working device, while holding the output to the real device de-energized, for use in testing or operator training.
- Monitors I/O communication status and alarms on an I/O fault. Optionally 'sheds' to the de-energized state on an I/O fault condition.
- Provides an 'Available' status when in Program command source and operating normally for use by automation logic to determine if the logic can manipulate the device.
- Operates in Operator, Program, External, Override, Maintenance, and Hand command sources.

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_DOut_4.10.00_AOIL5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Operations

This section describes the primary operations for this Add-On Instruction.

IMPORTANT See [Appendix A](#) for Command Sources and Simulation types.

Alarms

This instruction uses the following alarms, which are implemented by using embedded P_Alarm and P_Gate Add-On Instructions.

Table 5 - P_PDOut Alarms

Alarm Name	P_Alarm Name	P_Gate Name	Description
Interlock Trip	IntlkTrip	None	Raised when an interlock 'not OK' condition causes the device to transition from the On state or a pulsing state to the Off state. If interlocks are not bypassed, a bypassable interlock or a non-bypassable interlock 'not OK' condition initiates an interlock trip. If interlocks are bypassed, only a non-bypassable interlock 'not OK' condition initiates an interlock trip.
I/O Fault	IOFault	None	Raised when the Inp_IOFault input is true. This input is used to indicate to the instruction that a communication failure has occurred for its I/O. If the I/O Fault is configured as a shed fault, the device is commanded Off and cannot be commanded to another state until reset.
Off Fail	OffFail	None	Raised when the device is commanded Off, but the device feedback does not confirm that the device is actually Off within the configured failure time (Cfg_OffFailT).


Table 5 - P_PDOut Alarms

Alarm Name	P_Alarm Name	P_Gate Name	Description
On Fail	OnFail	None	Raised when the device is commanded On, but the device feedback does not confirm that the device is actually On within the configured failure time (Cfg_OnFailT). If the failure is configured as a shed fault, the device is commanded Off and cannot be commanded On until reset.

Simulation

Simulation in P_DOut de-energizes the output and simulates providing feedback of a working device. You can test the operation of the instruction under controlled conditions.

You must set the Inp_Sim parameter in the controller to '1' to enable simulation.

The Simulation icon  is displayed at the top left of the Operator faceplate indicating the device is in simulation.

You can use Cfg_SimFdbkT to delay the echo of the On/Off status of the device.

When you have finished in simulation, set the Inp_Sim parameter in the controller to '0' to return to normal operation.

Execution

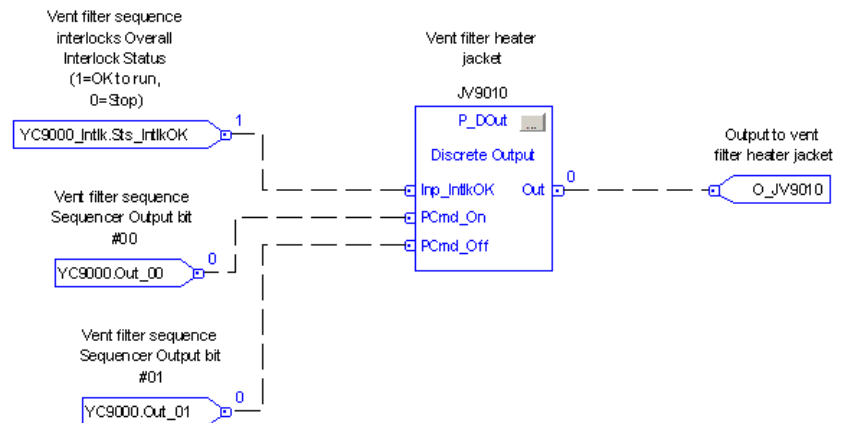
The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	Processing for EnableIn False (false rung) is handled the same as if the device were taken out of service by Command. The device outputs are de-energized and the device is shown as Program Out of Service on the HMI. All alarms are cleared.
Powerup (prescan, first scan)	On Prescan, any Commands received before First Scan is discarded. The device is de-energized. On first scan, the device is treated as if it were returning from Hand command source: the instruction state is set based on the feedback received from the device. Embedded P_CmdSrc and P_Alarm instructions are handled in accordance with their standard powerup procedures. See the P_CmdSrc and P-Alarm reference manuals for details.
Postscan	No SFC Postscan logic is provided.

Programming Example

This example uses the P_DOut instruction to control a heating jacket on a vent filter. The heating jacket is being used in this case to keep the vent filter dry when there is potential for condensate buildup.

In this example, the vent filter heater jacket does not provide the feedback on its status. In normal operating conditions, the vent filter heater jacket is being commanded on or off by the control sequence configured in the controller. If the operating status of the sequence is not OK, always command the vent filter off.



In this example, the controlling sequence issues commands to set the desired state of the vent filter heater. The parameter PCmd_On is triggered to command the vent filter heater on, and the parameter PCmd_Off is triggered to command the vent filter heater off.

The instruction is normally operated by sequence program logic. The P_CmdSrc (command source) instruction within the P_DOut instance must also be configured. The parameter CmdSrc.Cfg_ProgPwrUp is set to 1 so the instruction will power up with the Program command source selected. The parameter CmdSrc.Cfg_ProgNormal is set to 1 to indicate that the instruction will normally use the Program command source. If another command source is selected, the graphic symbol and faceplate will flag the selected source.

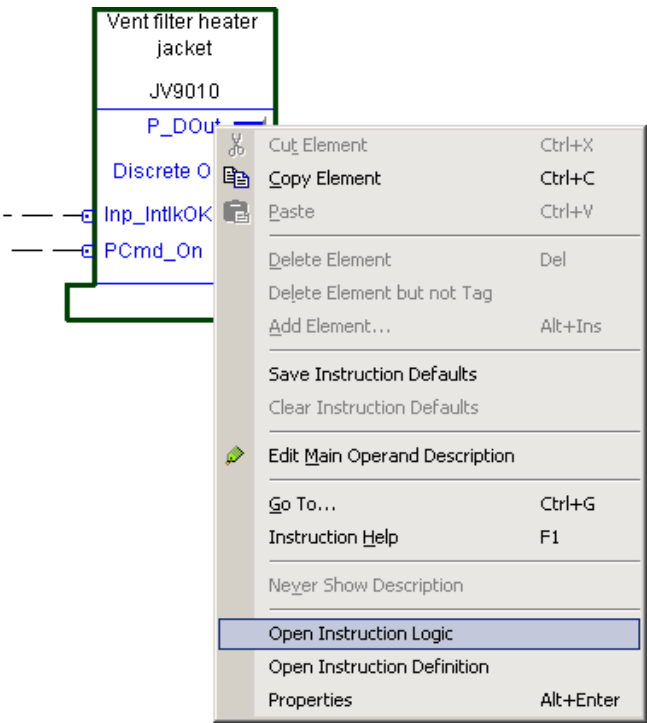
The status of the sequence is connected to the Inp_IntlkOK parameter so that the output to the vent filter heater jacket is always off when the skid is not operating properly, even if the instruction is not in Program mode.

The parameters Cfg_HasOnFdbk and Cfg_HasOffFdbk are both set to 0 to indicate that the vent filter heater jacket does not provide feedback on its status. The parameter Cfg_HasOnFailAlm, Cfg_HasOffFailAlm, Cfg_HasIntlkTripAlm, and Cfg_HasIOFaultAlm are all set to 0, indicating that no alarms are necessary for this device.

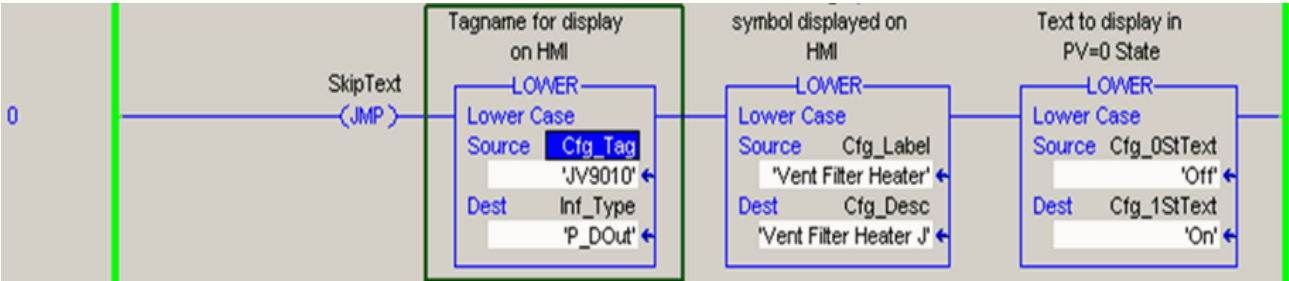
Lastly, configure the following local configuration tags to drive the text on the HMI faceplate. In this example, the vent filter P&ID tag is JV9010. In this example, they are set as follows:

Cfg_Tag: 'JV9010'
Cfg_Label: 'Vent Filter Heater'
Cfg_Desc: 'Vent Filter Heater Jacket'
Cfg_St0Text: 'Off'
Cfg_St1Text: 'On'

Local tags can be configured through the HMI faceplates or in Logix Designer application by opening the Instruction Logic of the Add-On Instruction instance and then selecting the string on the displayed rung.



The strings in local tags are shown on the first rung of the Add-On Instruction's Logic routine for your convenience.



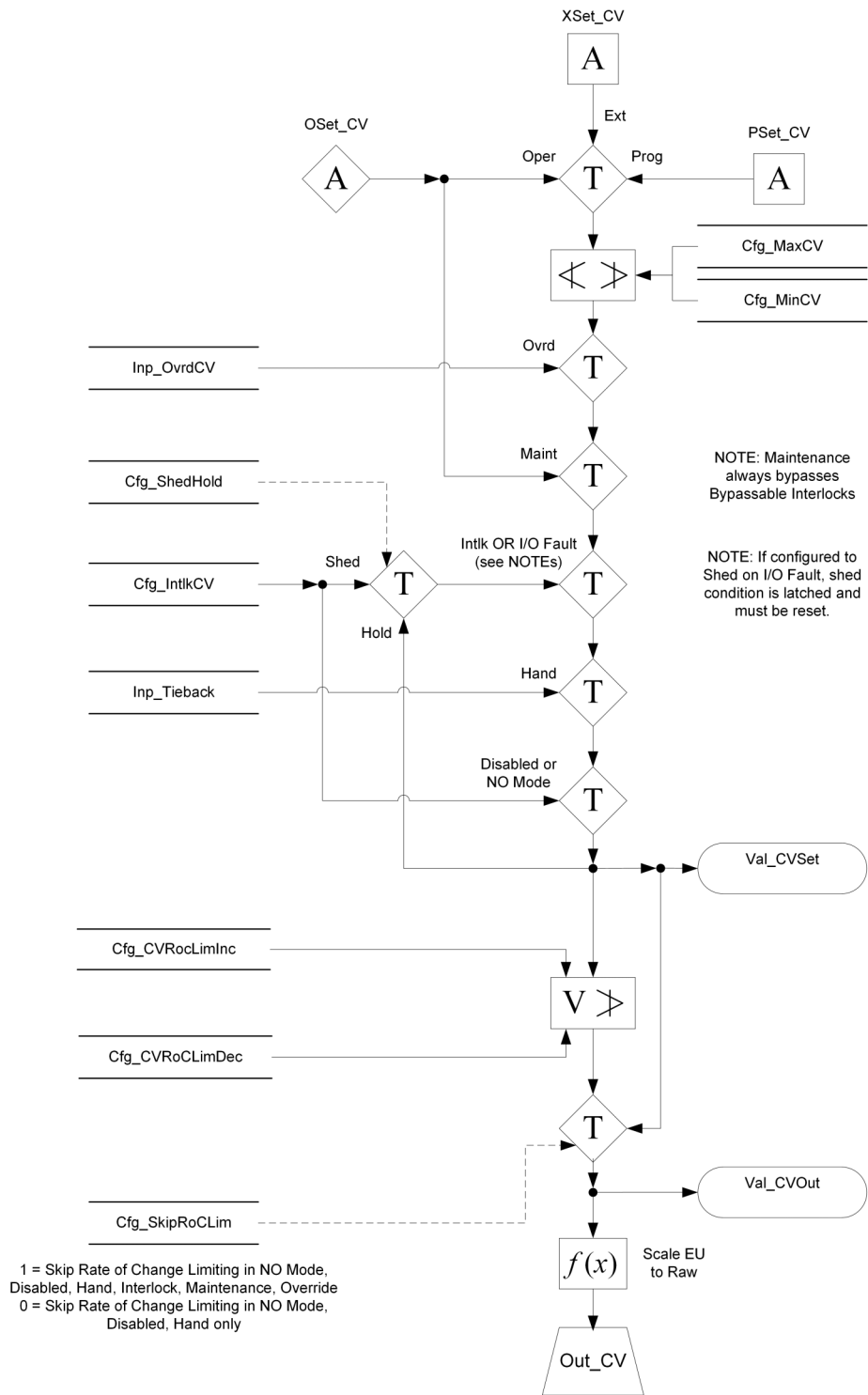
Analog Output (P_AOut)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_AOut (Analog Output) Add-On Instruction is used to manipulate an analog output to control a field device, such as a control valve or a motorized gate positioner. The output responds to an Operator (manual) or Program setting of the Controlled Variable (CV) signal.

The P_AOut instruction controls the analog output in a variety of modes (Operator, Program, Override, Maintenance, Hand), monitoring for fault conditions.

The diagram shows the functional characteristics of the P_AOut Add-On Instruction.



Functional Description

The P_AOut instruction provides the following capabilities:

- Monitors I/O fault input and raises an alarm on an I/O fault.
- Control of the Analog Output through the standard P_CmdSrc Add-On Instruction and modes.
- Ability for an operator or other Program logic to set an Analog Controlled Variable (CV, or output) to a specific value. The entered CV is scaled from engineering units to raw (output card) units.
- Interlocks (bypassable and non-bypassable) that are conditions that force the analog output to a specific configured (within tolerance) value or cause the output to hold its current value (configurable). Provides an alarm when an interlock causes the Analog Output CV to be changed. Provides maintenance personnel the capability to bypass the bypassable interlocks.
- Override CV input, which determines the CV in Override command source.
- Simulation capability, in which the output of the analog output is held at zero and I/O faults are ignored. The instruction can be manipulated as if a working analog output were present. This capability is often used for activities such as instruction testing and operator training.
- Increase and decrease rate of change limits (ramping) for the output that can be set by the operator or program. Provides a configurable limit for the maximum allowed rate of increase and for the maximum allowed rate of decrease.
- Tieback input (REAL) and a Hand command source request input (BOOL); when Hand command source is asserted, the CV is forced to follow the Tieback value.

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware software. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_AOut_4.10.00_AOIL5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Operations

This section describes the primary operations for this Add-On Instruction.

IMPORTANT See [Appendix A](#) for Command Sources and Simulation types.

Alarms

This instruction uses the following alarms, which are implemented by using embedded P_Alarm and P_Gate Add-On Instructions.


Alarm Name	P_Alarm Name	P_Gate Name	Description
Interlock Trip	IntlkTrip	None	Raised when an interlock 'not OK' condition causes the output CV to be changed to the configured Interlock CV value or held at its last value. If interlocks are not bypassed, a bypassable interlock or a non-bypassable interlock 'not OK' condition initiates an interlock trip. If interlocks are bypassed, only a non-bypassable interlock 'not OK' condition initiates an interlock trip.
I/O Fault	IOFault	None	Raised when the Inp_IOFault input is true. This input is used to indicate to the instruction that a communication failure has occurred for its I/O. If the I/O Fault is configured as a shed fault, the output CV is set to the configured Interlock CV or held at its last value until reset.

Parameters of the P_Alarm object can be accessed by using the following convention: [P_Alarm Name].[P_Alarm Parameter].

Simulation

Simulation in P_AOut simulates the requested CV, sets the Out_CV output to 0, and ignores any I/O Faults.

You must set the Inp_Sim parameter in the controller to '1' to enable simulation.

The Simulation icon  is displayed at the top left of the Operator faceplate indicating the device is in simulation.

When you have finished in simulation, set the Inp_Sim parameter in the controller to '0' to return to normal operation.

Execution

The following table explains the handling of instruction execution conditions.

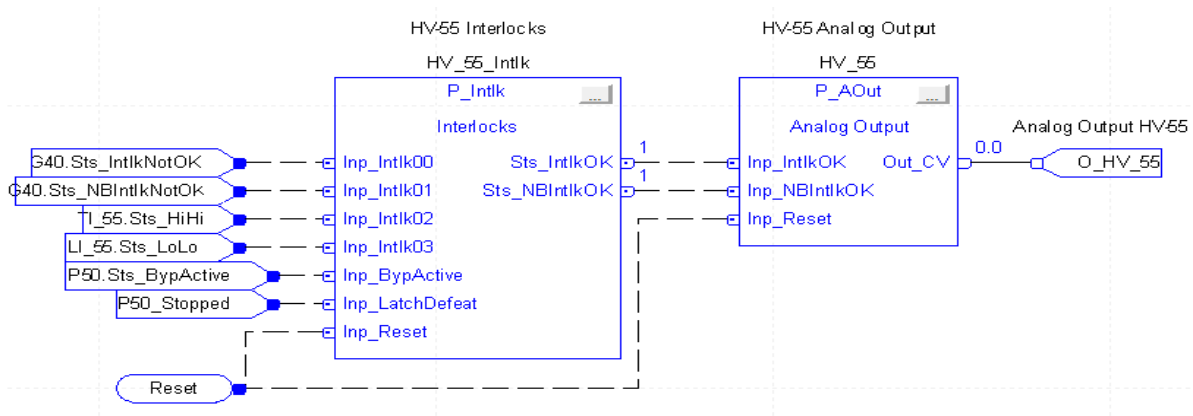
Condition	Description
EnableIn False (false rung)	Processing for EnableIn False (False Rung) is handled the same as if the Analog Output were taken out of service by command. The CV output is de-energized (zeroed) and the Analog Output instruction is shown as Program Out of Service on the HMI.
Powerup (prescan, first scan)	Processing of modes and alarms on prescan and powerup is handled by the embedded P_CmdSrc and P_Alarm instructions. See their manuals for details. On powerup, the analog output control is cleared; otherwise, all data remains in the state it was in at power down.
Postscan (SFC transition)	No SFC postscan logic is provided.

See the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#), for more information.

Programming Example

The following example provides a demonstration of using a P_AOut block to control a valve. During normal operation, the operator sets the valve position using P_AOut through the Add-On Instruction faceplate.

This example also includes interlock conditions using a P_Intlk block. The interlock conditions include upstream Group (G40) interlocks, plus low-low level and high-high temperature. The outputs of the interlock block are used as inputs by P_AOut to set the valve (Out_CV=O_HV_55) to an interlock position (for example, closed). This is done by setting the P_AOut configuration parameter Cfg_IntlkCV to 0.



Pressure/Temp. Compensated Flow (P_PTComp)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The Pressure/Temperature Compensated Flow (P_PTComp) Add-On Instruction is used to calculate a flow at standard temperature and pressure, essentially a mass flow rate, given a volumetric flow rate or differential pressure measurement. This instruction also requires measurements of the actual temperature and pressure of the flowing gas.

Functional Description

The P_PTComp Add-On Instruction is intended as a calculation function only, between other blocks, and no HMI components are provided. If a faceplate and/or alarms are needed, the calculated output from the instruction can be sent to a P_AIn (analog input) instruction for alarming and display.

This instruction includes the following capabilities:

- Takes as its primary input either a volumetric flow rate or a differential pressure across a flow element, such as an orifice plate or pitot tube. When a differential pressure is used, the P_PTComp instruction allows configuration of the volumetric flow rate for a given differential pressure.
- Accepts a temperature in common units (Fahrenheit or Celsius degrees) or in absolute units (Rankine degrees or Kelvins).
- Accepts a pressure in common units (PSIG, kPa Gauge, or MPa Gauge) or in absolute units (PSIA, kPa Absolute, MPa Absolute).

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware software. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_PTComp_4.10.00_AOI.L5X Add-On Instruction, must be imported into the controller project to use the instruction in the project. The service release number (boldfaced) can change as service revisions are created.

Operations

This section describes the primary operations for this Add-On Instruction.

Alarms

The P_PTComp Add-On Instruction does not generate any alarms.

To provide High-High, High, Low, and/or Low-Low threshold alarms for any of the variables, use a P_AIn Analog Input instruction for each such variable. These variables include volumetric flow or differential pressure, temperature, pressure, or the calculated Flow at Standard Conditions.

Simulation

The P_PTComp Add-On Instruction does not have simulation capability.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	No EnableInFalse logic is provided. The Instruction maintains its last state when EnableIn is false.
Powerup (prescan, first scan)	No Pre-scan or First Scan logic is provided. The P_PTComp instruction simply performs its calculation every scan when EnableIn is true.
Postscan	No SFC Postscan logic is provided.

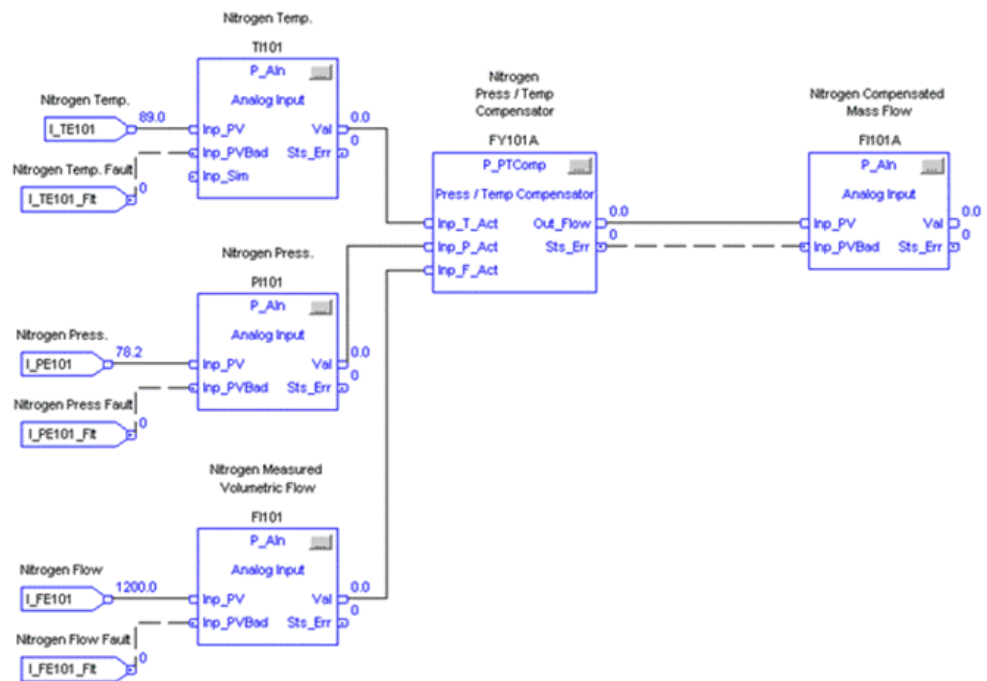
See the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#), for more information.

Programming Example

This example uses the P_PTComp instruction to determine the flow rate of compressed nitrogen at a standard pressure and flow. This can provide a more accurate measurement for custody transfer or control calculations where there is variability in environmental conditions and the flow transmitter is not capable of performing the compensation.

In this case, we have a measurement of flow from a dp-transmitter. The transmitter is providing the controller with a value that has been scaled to volumetric flow but not compensated for environmental temperature and pressure. We also have temperature and pressure measurements from where the

flow is measured. In this example, the desired standard pressure and flow is 0 psig and 15 °C.



The measured temperature, pressure, and flow are connected into the P_PTComp instruction to the inputs Inp_T_Act, Inp_P_Act, and Inp_F_Act. In this example, these values are in units of degrees C, psig, and m³/hr.

Set `Cfg_T_Std` and `Cfg_P_Std` to 15 and 0, respectively, for the desired standard temperature and pressure. `Cfg_T_Offset` is left at its default of 273.15 to represent 0 °C in absolute units K (if using Fahrenheit, set this to 459.67 °F). `Cfg_P_Offset` is left at its default of 14.696 defining the value of 0 psi gauge pressure in absolute pressure. `Cfg_UseDP` is left at its default of 0, indicating we are using `Inp_F_Act` as the flow input as the flow transmitter is providing flow in volumetric units.

The output of `P_PTComp` is then connected to a `P_AIn` Instruction. The output is a compensated volumetric flow at standard temperature and pressure. The `P_AIn` could scale this flow to mass flow if desired.

The local configuration tags `Cfg_Desc`, `Cfg_Label`, and `Cfg_Tag` are not required to be set. The `P_PTComp` instruction does not include visualization elements (global objects or faceplates). However, these string parameters are provided for use in custom visualization elements if desired.

Tank Strapping Table (P_StrapTbl)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_StrapTbl (Tank Strapping Table) Add-On Instruction calculates the volume of product in an upright cylindrical tank, given the level of the product and the tank calibration table. This instruction can optionally compensate for free water at the bottom of the tank (given a product/water interface level) or for thermal expansion of the tank shell (given the coefficient of linear expansion of the shell material and product and ambient temperatures).

This instruction also can optionally compensate for a floating tank roof if the product density is provided.

Functional Description

The P_StrapTbl instruction is intended only as a calculation function, between other blocks, and so no HMI components are provided.

For a faceplate or alarm, send the calculated corrected volume to a P_AIn (Analog Input) instruction.

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware software. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The Add-On Instruction import, P_StrapTbl_4.10.00_AOIL5X, must be imported into the controller project to use the instruction in the project. The service release number (boldfaced) can change as service revisions are created.

Operations

The P_StrapTbl Add-On Instruction uses tank calibration data and a tank level measurement to calculate tank volume. Tank calibration data can be obtained from the tank manufacturer or design firm, or determined through calibration. Example calibration methods are provided by the American Petroleum Institute's

(API) Manual of Petroleum Measurement Standards (MPMS) section 2.2A or 2.2B.

This instruction performs its calculations by using the same methods described in API MPMS Section 12.1 Part 1. The instruction calculates the following items:

- Total Observed Volume (TOV)
- Free Water Volume (FW)
- Correction for Temperature of Tank Shell (CTSh)
- Floating Roof Adjustment (FRA)
- Gross Observed Volume (GOV, the primary output of this instruction)

All calculations require the overall level input and the tank calibration table ('strapping table'). The FW calculation is a specific requirement when calculating petroleum storage to adjust for free water content and requires an additional level signal for the product/water interface.

The CTSh calculation compensates for thermal expansion of the storage tank. If used, this calculation requires these measurements and configuration settings:

- Product temperature and the ambient temperature
- Configuration of the tank shell coefficient of linear expansion (fraction per degree)
- Calibration temperature for the tank calibration table
- Configuration constant for weighting the two measured temperatures (a reasonable default is provided)

The FRA calculation compensates the level measurement for displacement caused by a floating roof. This calculation requires a measurement or input of the product actual density and additional configuration data about the floating roof.

The final GOV calculation is provided without correction for product density and temperature (other than the shell temperature above) or included sediment and water. Those calculations depend on the product, not just the tank calibration table, and are beyond the scope of this instruction.

Alarms

The P_StrapTbl Add-On Instruction only performs calculations and does not have any alarms. The instruction does not contain an embedded P_Alarm instruction.

To provide High-High, High, Low, and/or Low-Low threshold alarms for the input or output of P_StrapTbl, use the P_AIn Analog Input instruction.

Simulation

The P_StrapTbl Add-On Instruction does not have simulation capability.

Use the simulation capability of associated analog input instructions to simulate level or temperature inputs or volume outputs.

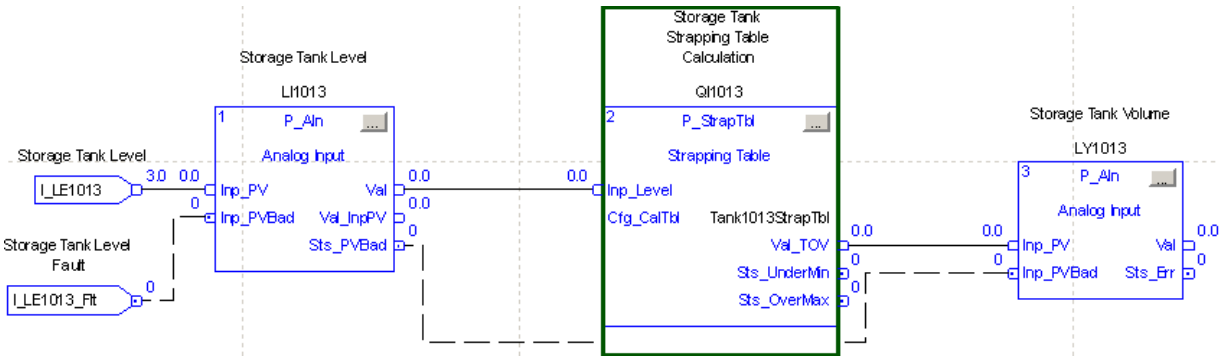
Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	No EnableInFalse logic is provided. The instruction maintains its last state when EnableIn is false.
Powerup (prescan, first scan)	No Pre-scan or First Scan logic is provided. The P_StrapTbl instruction simply performs its calculation every scan when EnableIn is true.
Postscan	No SFC Postscan logic is provided.

Programming Example

This example uses the P_StrapTbl instruction to calculate the volume of product in a storage tank based on the measured storage tank level and storage tank strapping table information. In this example, there is no floating roof so there is no compensation for displacement. There are no adjustments based on temperature to account for thermal expansion of the tank.



The measured storage tank level is connected into the P_StrapTbl instruction by using the input Inp_Level. In this example, the level is reported in units of feet.

The storage tank is a 4 ft tall tank, and strapping table information has been provided by the tank vendor. Strapping tables often list data in terms of major and minor units. In this example, data has been provided at 6-in increments. The vendor-provided strapping table has nine rows and looks like the following table.

Level (ft-in.)	Volume (barrels)
0-00	3.1
0-06	136.6
1-00	264.2
1-06	402.7
2-00	541.4
2-06	692.7

Level (ft-in.)	Volume (barrels)
3-00	844.1
3-06	990.8
4-00	1137.5

To store the strapping table information in the controller, the tag Tank1013StrapTbl is created as type P_StrapTblRow[9] (a nine-element array of strapping table information).

The parameter Cfg_MinorPerMajor is left at its default of 12, so that the instruction can convert the input (feet) to major and minor units (feet and inches). This allows the strapping table to be configured by using the same major and minor units as provided by the vendor.

Table 6 - Strapping Table Calculation Examples

	.Major	.Minor	.Volume
Tank1013StrapTbl[0]	0	0	3.1
Tank1013StrapTbl[1]	0	6	136.6
Tank1013StrapTbl[2]	1	0	264.2
Tank1013StrapTbl[3]	1	6	402.7
Tank1013StrapTbl[4]	2	0	541.4
Tank1013StrapTbl[5]	2	6	692.7
Tank1013StrapTbl[6]	3	0	844.1
Tank1013StrapTbl[7]	3	6	990.8
Tank1013StrapTbl[8]	4	0	1137.5

The InOut tag Cfg_CalTbl of the P_StrapTbl instruction is modified to point to the new array Tank1013StrapTbl to provide the instruction with the strapping table information. The output of P_StrapTbl is then connected to another P_AIn instruction. The output is the calculated volume of the storage tank.

The local configuration tags Cfg_Desc, Cfg_Label, and Cfg_Tag are not required to be set. The P_StrapTbl instruction does not include visualization elements (global objects or faceplates). However, these string parameters are provided for use in custom visualization elements, if desired.

HART Analog Input (P_AInHART)

All relevant information has been placed in publication [PROCES-RM010](#), Rockwell Automation Library of Process Objects: HART Modules for PlantPAx DCS.

HART Analog Output (P_AOutHART)

All relevant information has been placed in publication [PROCES-RM010](#), Rockwell Automation Library of Process Objects: HART Modules for PlantPAx DCS.

Regulatory and Procedural Control

Purpose

This chapter is for the operation of the Add-on Instructions. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

Library objects in this section comprise two groups of Advanced Process Control: regulatory and procedural.

- Regulatory control focuses on the process variables (levels, flows, temperatures, pressures, and so on). The control is designed to improve loops that perform poorly and automate loops that are typically run in manual by the operator. These loops include techniques such as ratio, feed-forward, cascade, and split-range control.
- Procedural control focuses on the product being manufactured (cement, polyethylene, ethanol, paper, and so on). The objects control the various product specifications and parameters via a series of discrete sequential actions.

[Table 7](#) describes the objects in this group, including when to use and not to use each one.

Table 7 - Regulatory Control

Process Object	Description	When to Use	When Not to Use
Proportional + Integral + Derivative Enhanced (P_PIDE)	This instruction provides the functionality of the PIDE built-in instruction for PID loop control and additional alarm status information. Use this instruction when you plan to use the PIDE for loop control and provide visualization to the operator.	<ul style="list-style-type: none"> • Provides additional context for display, including a description, label, tag, and engineering units. • Provides P_CmdSrc (command source) object for control. • Links for the P_Intlk (interlocks) instruction for interlocking. • Power-up Setpoint, Output, and Loop mode settings. 	
Analog Fanout (P_Fanout)	This instruction fans one 'primary' analog output signal out to multiple 'secondary' users or devices. Each secondary output has configurable gain, offset, and clamping limits. The instruction applies minimum and maximum clamping limits to each output (secondary) CV.	<ul style="list-style-type: none"> • To fan the output of a PID loop, or a P_AOut (Analog Output) Add-On Instruction used as a manual loading station, to multiple valves, drives, or other devices. • You have 2...8 devices that are driven by the loop or output. • You have valves, drives, or other output devices that react over different ranges of the PID or P_AOut output, such as a 'Split-Range' control strategy. • You want to initialize the primary output signal when all secondaries are requesting initialization. 	<ul style="list-style-type: none"> • Have only one output device. Use the P_AOut instruction or other output instruction instead. The P_Fanout capabilities are targeted to control strategies where there is a need to have one loop or station drive multiple devices. • Have multiple PID loops and one output device that uses the highest or lowest of the PID loop outputs (high-select or low-select strategy). Use the ESEL built-in instruction, the P_HiLoSel instruction, or other high-select/low-select logic.

Table 7 - Regulatory Control

Process Object	Description	When to Use	When Not to Use
High or Low Selector (P_HiLoSel)	<p>This instruction selects the lowest of the (up to 6) incoming CVs or the highest of the incoming CVs and outputs the value.</p> <p>The unselected CVs are flagged to track the selected CV. The tracking value can optionally be offset by an amount equal to the upstream PID/PIDE Gain times Error to avoid problems with ever-decreasing (if low-select) or ever-increasing (if high-select) output.</p>	<ul style="list-style-type: none"> To implement an Override Select control strategy. This strategy controls a primary process variable while other process variables can override the output to the final control element to avoid exceeding constraints. <ul style="list-style-type: none"> For example, a pump station uses a PID loop with a variable speed drive on the main line pump as the final control element to control discharge pressure (the primary PV). But additional PID controls are provided that reduce the speed of the pump if the pump motor current is too high or if pump suction pressure is too low. 	<ul style="list-style-type: none"> Have multiple interacting loops each with their own final control element. Use the Model Predictive Control built-in instructions instead. Have one process variable control loop with multiple final control elements. Use the P_Fanout Add-On Instruction or the SRTP (split-range/time proportional) built-in instruction instead.
Deadband Controller (P_DBC)	<p>This instruction provides control of a process variable within limits by using one or two discrete outputs. A deadband controller is also known as a 'bang-bang' or 'on-off' controller.</p>	<ul style="list-style-type: none"> Control an analog process variable (PV), such as temperature, level or pressure, between upper and lower control limits by triggering one or two discrete outputs. The outputs typically energize and de-energize equipment that increases or decreases the process variable. 	<ul style="list-style-type: none"> To control an analog process variable by using an analog final control element, such as a modulating control valve or a variable speed motor drive, use the P_PIDE Add-On Instruction, the PIDE function block, PID ladder instruction, or the IMC, CC, or MMC function blocks for model-based control of the PV.
Sequencer Object (P_Seq)	<p>This instruction is a controller-based step sequencing solution that reduces engineering time by automating common operator procedures.</p> <p>The step-by-step configuration makes it easy to adjust procedures directly from the HMI displays without having to create or modify custom code in the controller. The Sequencer can be employed in simple and complex sequences without costly re-engineering and testing. You add, delete, or modify steps that are used accomplish the objective of the sequence.</p>	<ul style="list-style-type: none"> To implement a procedure to operate equipment in a prescribed order (open valve, start pump, and so forth). The Sequencer is intended for basic sequencing that is typical of control and equipment implementation modules (as defined in ISA-TRI06.00.01). The instruction can be used at any level and in any application where its functionality is appropriate. 	<ul style="list-style-type: none"> Implement a batch phase, such as material addition, agitation, transfer, and so forth, where a hold, restart, or reset of logic is required. Use the Phase Manager capability of Logix Controller instead. Need sophisticated sequential function chart (SFC) procedures, such as simultaneous threads and multi-selection branches.
Dosing (P_Dose)	<p>The P_Dose Add-On Instruction controls ingredient addition to measure the quantity of ingredient that is being added. The instruction can be used with a flowmeter or weigh scale.</p> <p>The flowmeter can be:</p> <ul style="list-style-type: none"> Analog flowmeter (signal proportional to flow) Pulse generating flowmeter (pulse count proportional to quantity delivered) Digital flowmeter providing flow rate or quantity (totalized flow) information. <p>The weigh scale can be on the receiving vessel (gain in weight) or on the sourcing vessel (loss in weight). The weigh scale can be connected via an analog input, device network, or other connection.</p>	<ul style="list-style-type: none"> Want to control basic dosing (ingredient addition) with basic features, such as bulk/dribble rate selection, preact, automatic preact adjustment, and the ability to start, pause, and resume flow. Measuring the quantity of ingredient added by using a flowmeter. The flowmeter can provide an analog flow rate, an analog quantity (total), or a pulse count with rollover. Measuring the quantity of ingredient added to a destination vessel (gain in weight) or the quantity that is transferred from a source vessel (loss in weight) by using a weigh scale. The scale provides a weight value and can be interfaced via an analog input card, network, or other means. 	<ul style="list-style-type: none"> This instruction does not include capability for controlled-rate addition, such as ratio control, digital blending, or precision blending. Contact your Rockwell Automation representative for a blending solution. Need a totalizer (integrator) only. Use the built-in TOT instruction instead. Need to display a weigh scale's weight or generate high and low weight alarms. Use the P_AIn Analog Input instruction instead.

Table 7 - Regulatory Control

Process Object	Description	When to Use	When Not to Use
Lead/Lag/Standby Motor Group (P_LLS)	This instruction provides control of a parallel group of motors. Such groups are commonly used for a group of pumps that maintain pressure on a header despite wide changes in demand. For example, municipal-scale or plant-scale water distribution.	<ul style="list-style-type: none"> • Need ability for the Operator or Program to enter a 'demand', the number of motors to run. • Need a configurable ability to stop the last started motor or the first started motor (first-on-last-off or last-on-last-off). • Need configurable delay between starts and configurable delay between stops. Starts or stops motors as required to meet the entered demand. • To identify (and optionally alarm) when there are not enough motors available to start for the given demand to be met. 	<ul style="list-style-type: none"> •

Proportional + Integral +Derivative Enhanced (P_PIDE)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_PIDE (Proportional + Integral + Derivative Enhanced) Add-On Instruction provides the functionality of the Studio 5000 Logix Designer® PIDE function block with a user experience consistent with the rest of the Rockwell Automation Library of Process Objects.

Functional Description

The primary operations of the P_PIDE Add-On Instructions and its faceplate include the following:

- All the functions of the PIDE built-in instruction for PID loop control
- Process Library alarm objects for deviation alarms, and additional alarm status information and functionality, including limits, deadbands, and severities
- Additional context for display, including a description, label, tag, and engineering units
- P_CmdSrc (command source) object for control
- Links for the P_Intlk (interlocks) instruction for interlocking
- Power-up Setpoint, Output, and Loop mode settings
- Setpoint ramping over a specified time.

Autotune

You must have a license to edit the autotune tag entry field on the PIDE instruction. Complete these steps to enable the functionality.

1. Open the Logic routine of the 'P_PIDE_only' Add-On Instruction.
2. To set the autotune tag to 'Ref_Autotune', edit the function block diagram.
3. Save your changes and download to your controller.

Once this change has been made, the outer P_PIDE instruction automatically checks on power-up for response from the Autotune function. The Autotune button is automatically enabled on the faceplate.

This function is a supported end-user/solution-provider enhancement that does not void technical support.

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_PIDE_4.10.00_AOIL5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Operations

This section describes the primary operations for this Add-On Instruction.

IMPORTANT See [Appendix A](#) for Command Source and Simulation types.

Alarms

This instruction uses the following alarms, which are implemented by using embedded P_Alarm and P_Gate Add-On Instructions.

Table 8 - P_PIDE Alarms

Alarm Name	P_Alarm Name	P_Gate Name	Description
Fail	Fail	None	<p>Raised when the internal PIDE instruction reports an Instruction Fault. The PIDE instruction reports an Instruction Fault under any of the following conditions:</p> <ul style="list-style-type: none"> • Setpoint value has bad quality or is invalid or cascade setpoint communication has faulted. • PV value has bad quality or is invalid or PV engineering units scaling configuration is invalid. • CV value has bad quality or is invalid or CV communication is faulted. • The Feed forward input value is invalid. • The Hand feedback input value is invalid or has bad quality.
Interlock Trip	IntlkTrip	None	<p>Raised when an interlock 'not OK' condition causes the output CV to be changed to the configured Interlock CV value or held at its last value. If interlocks are not bypassed, a bypassable interlock or a non-bypassable interlock 'not OK' condition initiates an interlock trip. If interlocks are bypassed, only a non-bypassable interlock 'not OK' condition initiates an interlock trip.</p>
High Deviation	HiDev	HiDevGate	Raised when the amount by which the PV exceeds the setpoint is above the High Deviation threshold. The threshold, deadband, gating, and timing are set in configuration.
High-High Deviation	HiHiDev	HiHiDevGate	Raised when the amount by which the PV exceeds the setpoint is above the High-High Deviation threshold. The threshold, deadband, gating, and timing are set in configuration.
Low Deviation	LoDev	LoDevGate	Raised when the amount by which the PV exceeds the setpoint is below the Low Deviation threshold. (Since the threshold is a negative number, this reading is the amount the PV falls below the setpoint or reference.) The threshold, deadband, gating, and timing are set in configuration.
Low-Low Deviation	LoLoDev	LoLoDevGate	Raised when the amount by which the PV exceeds the setpoint is below the Low-Low Deviation threshold. (Since the threshold is a negative number, this reading is the amount the PV falls below the setpoint or reference.) The threshold, deadband, gating, and timing are set in configuration.

Parameters of the P_Alarm object can be accessed by using the following convention: [P_Alarm Name].[P_Alarm Parameter].

Simulation

The P_PIDE Add-On Instruction does not have Simulation capability.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	The instruction command source is shown as Program Out of Service. The loop status is shown as disabled. The loop CV is set to the configured interlock CV value.
Powerup (prescan, first scan)	Received commands are cleared. The loop is initialized with the powerup Loop mode, CV, and SP.
Postscan (SFC Transition)	No SFC postscan logic is provided.

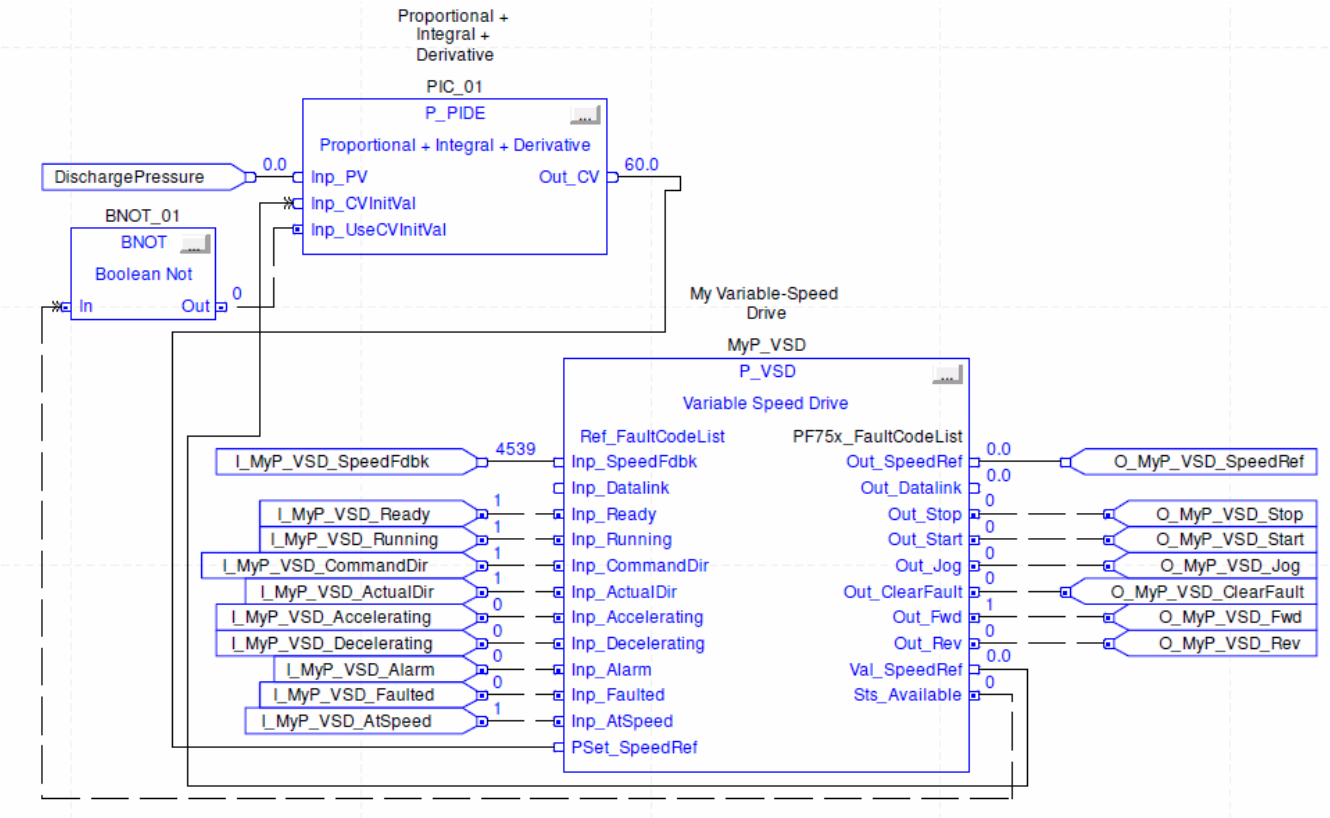
See the Logix 5000™ Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#), for more information.

Programming Example

An example showing P_PIDE with P_VSD is shown below.

The output of the P_PIDE block (Out_CV) is used as the input to the P_VSD block (PSet_SpeedRef). The P_VSD output Sts_Available is True when the drive is available to be controlled by the program. When this value is False (the drive is not in program mode), the P_PIDE input Inp_UseCVInitVal is set to True, forcing the P_PIDE block to initialize its CV value to Inp_CVInitVal.

Inp_CVInitVal is connected for the output Val_SpeedRef (speed target to the drive).



Analog Fanout (P_Fanout)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The Analog Fanout (P_Fanout) Add-On Instruction fans one 'primary' analog output signal out to multiple 'secondary' users or devices. Each secondary output has configurable gain, offset, and clamping limits.

Functional Description

The Analog Fanout instruction provides the following capabilities:

- Receives an input CV (controlled variable) from a primary PID loop or analog output.
- Applies rate-of-change limiting to the input signal.
- Calculates outputs for up to eight secondary devices. Each secondary has its own ratio (slope) and offset (intercept) from the rate-limited primary input. The ratios and offsets are configured values.
- Applies minimum and maximum clamping limits to each output (secondary) CV.
- Provides for initialization of each of its secondary CV outputs based on a request bit and a requested value from the secondary. When a particular output CV comes out of initialization, it is ramped from the initialization value to its calculated value by using a configured 'Takeup' Rate.

- Provides for initialization of the primary when all secondaries have requested initialization. The initialization value sent to the primary can be a fixed (configured) value or a calculated value based on the CV1 (Output 1) requested initialization value, accounting for the CV1 gain and offset. Thus CV1 is the 'priority' output.

TIP If you are using the P_Fanout Add-On Instruction in a split-range strategy (its default configuration), use CV1 for the 'safe' part of the range (for example, a chilled water valve) and CV2 for the 'unsafe' part of the range (for example, a steam valve). If both CV1 and CV2 request initialization, the loop (primary) is initialized based on the requested value from CV1 and set to a value in the cooling range.

For example, a P_Fanout Add-On Instruction is configured to use input range 0...50% as 100...0% open on the cooling valve on CV1, and input range 50...100% as 0...100% on the heating valve on CV2. If both valves request initialization, the P_Fanout Add-On Instruction uses the CV1 initialization value and requests the primary to initialize in the 0...50% range, the cooling side.

If the heating valve is used as CV1, the initialization is always in the heating range of the primary CV. In many split-range applications, it is a requirement to initialize or fail in the cooling range (for example, 0...50% output, for 100...0% cooling and always 0% heating).

The default configuration of the P_Fanout instruction provides this cooling (CV1) and heating (CV2) setup, with CV3...CV8 not used.

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_Fanout_4.10.00_AOIL5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Operations

This section describes the primary operations for this Add-On Instruction.

IMPORTANT See [Appendix A](#) for Command Source and Simulation types.

Alarms

The P_Fanout instruction does not provide any alarms. The instruction does provide Status bits that identify if the input CV or any particular output CV is being limited. The instruction also provides Status bits if any input value (input CV or any of the individual CV initialization values) is Infinite (Inf) or Not a Number (NaN).

Simulation

The P_Fanout Add-On Instruction does not have simulation capability.

Execution

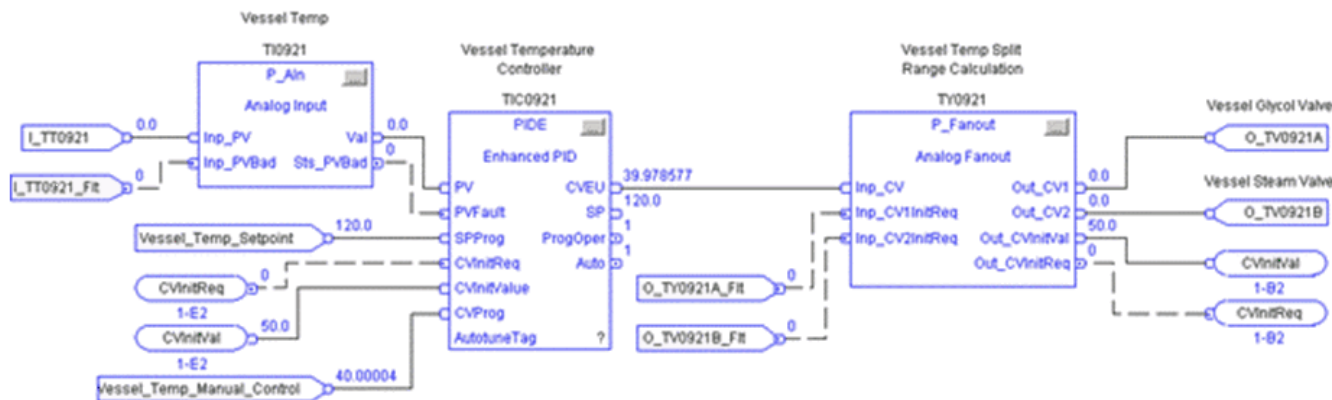
The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	The instruction is kept in its last state.
Powerup (prescan, first scan)	The CV Rate limiter is set to initialize at the first valid CV received.
Postscan	No SFC Postscan logic is provided.

See the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#), for more information.

Programming Example

This example uses the P_Fanout instruction to implement a split range PID control strategy to control temperature of a processing vessel. In this example, the heat exchanger to the vessel jacket is fed by either a steam valve to heat or a glycol valve to cool. A single PID controller is used to control temperature. It is assumed that the relative process gain between each valve and the temperature is the same.



To connect the PIDE instruction to the P_Fanout instruction, the PIDE output (CVEU) is connected to the input (Inp_CV) of P_Fanout. P_Fanout outputs Out_CVInitVal and Out_CVInitReq are connected to PIDE inputs CVInitReq and CVInitValue to be sure of proper initialization of the PIDE loop if there are issues with either valve.

Cfg_HasCV2 is set to 1 to indicate P_Fanout connects to two outputs. Cfg_CV1RatioSrc, Cfg_CV1OffsetSrc, Cfg_CV2RatioSrc, and Cfg_CV2OffsetSrc are all left at 0 to indicate that the scaling used to calculate the valve outputs is configured and not dynamically set by the operator or program.

To handle initialization, Cfg_FixedInitVal is set to 50 so that the PIDE instruction initializes with both valves closed when initialization is requested. Cfg_UseFixedInit is set to 1 to indicate the fixed initialization value is to be used instead of the feedback from the glycol valve.

To properly scale the two outputs, the scaling configuration values are set as follows:

Cfg_CV1Ratio:	-2.04
Cfg_CV1Offset:	100
Cfg_CV1Min:	0
Cfg_CV1Max:	100
Cfg_CV2Ratio:	2.04
Cfg_CV2Offset:	-103

Cfg_CV2Min: 0
Cfg_CV2Max: 100

These values cause a 50% output on the vessel temperature controller to command both the glycol and the steam valve closed (0%). As the PIDE output approaches 0%, the glycol valve opens (approach 100%). As the PIDE output approaches 100%, the steam valve opens (approach 100%). These settings create a little deadband around 50% where neither valve opens to prevent chattering between glycol and steam to prevent excessive wear on the heat exchanger.

P_Fanout outputs Out_CV1 and Out_CV2 are connected to the outputs to the glycol and steam valves. Valve status information is brought in through inputs Inp_CV1InitReq and Inp_CV2InitReq to be sure that the control loop initializes if there is a problem with a valve. Based on the settings above, initialization commands both valves closed.

Lastly, the following local configuration tags are configured to drive the text on the HMI global object and faceplate. In this example, they are set as follows:

Cfg_Tag: TY0921
Cfg_Label: Vessel 0900 Split Range
Cfg_Desc: Vessel Split Range Calculation
Cfg_CV1_Label: Glycol Valve
Cfg_CV2_Label: Steam Valve
Cfg_CV1_EU: %
Cfg_CV2_EU: %

High or Low Selector (P_HiLoSel)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

This instruction sets its output to the lowest or highest of the incoming CVs. For example, three PID controls feed a P_HiLoSel instruction that is configured to select the lowest of the three PID outputs as the speed reference for a drive. In normal operation, the discharge pressure PID has control, and the other PIDs track the output of the discharge pressure loop. When motor current exceeds its setpoint, or if suction pressure falls below its setpoint, the constraint being exceeded takes control to help prevent motor overcurrent or pump cavitation.

The P_HiLoSel (High or Low Selector Add-On Instruction) selects the lowest of the (up to 6) incoming CVs (if Cfg_HiLoSel = 0), or the highest of the incoming CVs (if Cfg_HiLoSel = 1) and outputs it (Out_CV).

The unselected CVs are flagged to track the selected CV.

The tracking value can optionally be offset by an amount equal to the upstream PID/PIDE Gain * Error to avoid problems with ever-decreasing (if Low-Select) or ever-increasing (if High-Select) output.

IMPORTANT Each CV input must come from the PIDE 'CV' (in percent); each proportional gain input must come from the PIDE 'PGain' parameter; and each error input must come from the PIDE 'EPercent' parameter.

Scaling of the output of this block to CVEU can be done by a downstream P_ValveC or P_AOut block. This block also supports initialization from a downstream block; the initialization is forwarded (with offset, if so configured) to upstream blocks.

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_HiLoSel_4.10.00_AOIL5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Operations

This section describes the primary operations for this Add-On Instruction.

Alarms

This instruction does not generate any alarms. Alarms are provided by upstream (P_PIDE) and downstream (P_ValveC, P_VSD, P_AOut) instructions as necessary.

Simulation

The P_HiLoSel Add-On Instruction does not have a Simulation capability.

Execution

The following table explains the handling of instruction execution conditions.

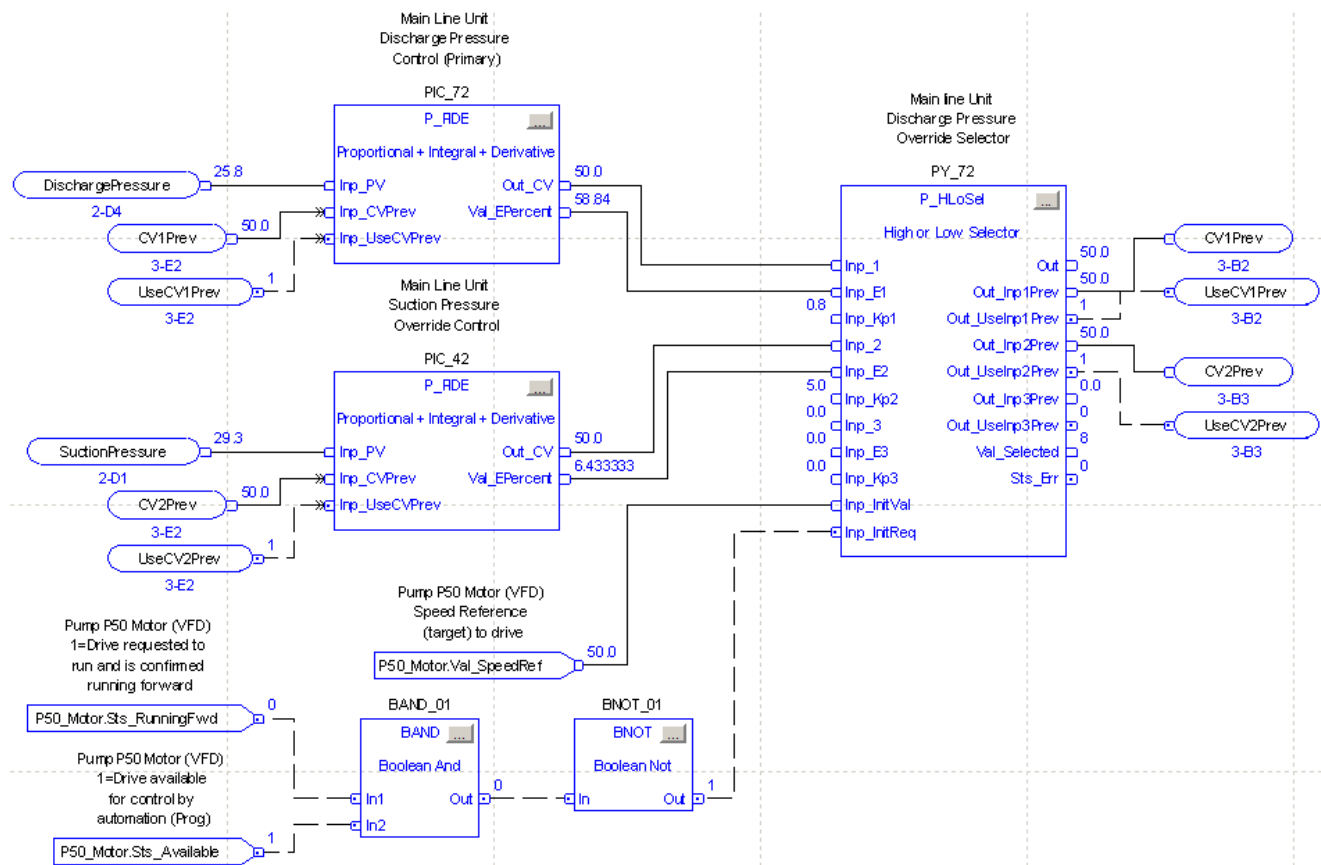
Condition	Description
EnableIn False (false rung)	No enableIn false logic is provided. If scanned in a Ladder or Function Block routine with the EnableIn input false, all values are held in their last states.
Powerup (prescan, first scan)	No prescan logic is provided.
Postscan (SFC transition)	No SFC postscan logic is provided.

See the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#), for more information.

Programming Example

The following example shows the P_HiLoSel in function block context and implements part of a pressure control strategy. In this case, two P_PIDE instructions are used as inputs for P_HiLoSel. The PIDE instructions are for Suction Pressure Override Control and Discharge Pressure Control. The P_PIDE output values Out_CV (CV to final control element) and Val_E (Loop Error) are used as inputs to P_HiLoSel.

This example also shows P_HiLoSel inputs for Initial Value (Inp_CVInitVal) and Initialization Required (Inp_CVInitReq). In this case, the Initial Value is taken from the speed reference to the pump motor drive. The Initialization Required flag is set based on the motor's running and availability status.



Deadband Controller (P_DBC)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_DBC (Deadband Controller) Add-On Instruction provides control of a process variable within limits by using one or two discrete outputs. A deadband controller is also known as a 'bang-bang' or 'on-off' controller.

Functional Description

The P_DBC instruction provides the following capabilities:

- A Raise output, which is activated when the PV is less than the entered Raise threshold.
- A Lower output, which is activated when the PV is greater than the entered Lower threshold.
- Q and Q-Not outputs. Q is set when the PV falls below the Raise threshold and cleared when the PV rises above the Lower threshold; Q-Not is the inverse of Q.
- High and Low Deviation alarms with configurable thresholds and deadbands. These alarms can provide notification that the PV is approaching an out-of-control condition.
- Alarms for High PV Rate of Change Increasing and High PV Rate of Change Decreasing. These alarms can provide notification that the PV is changing faster than expected.
- Operation in Manual and Automatic Loop Modes. In Automatic Loop Mode, the control algorithm triggers the outputs to keep the PV within limits. In Manual Loop Mode, the operator directly manipulates the Raise and Lower outputs from the HMI.
- Operation from Operator, Program, External, Override, and Maintenance command sources.

Required Files

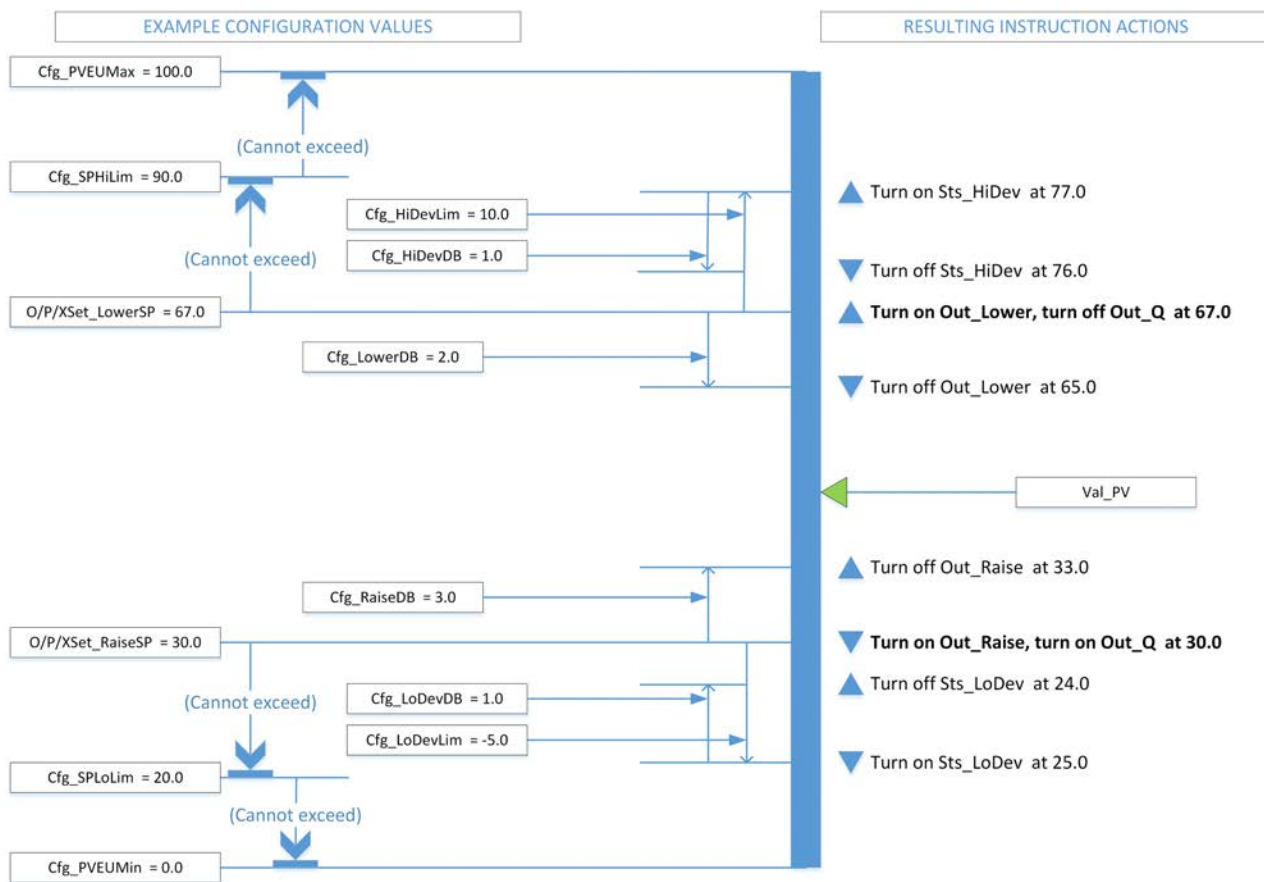
Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. The objects let you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_DBC_4.10.00_AOIL5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Operations

The image shows the reactions of the instruction to the value of PV as it increases or decreases.



Alarms

This instruction uses the following alarms, which are implemented by using embedded P_Alarm and P_Gate Add-On Instructions.

Alarm Name	P_Alarm Name	P_Gate Name	Description
High Deviation	HiDev	HiDevGate	Raised when the amount by which the PV exceeds the setpoint or reference is above the High Deviation threshold. The threshold, deadband, gating, and timing are set in configuration.
High Rate of Change (Decreasing)	HiRoCDec	HiRoCDesGate	PV rate of change that exceeds the High Rate of Change limit decreasing. Threshold, deadband, and severity are set in configuration.
High Rate of Change (Increasing)	HiRoCInc	HiRoCIncGate	PV rate of change that exceeds the High Rate of Change limit increasing. Threshold, deadband, and severity are set in configuration.
Low Deviation	LoDev	LoDevGate	Raised when the amount by which the PV exceeds the setpoint or reference is below the Low Deviation threshold. (Since the threshold is a negative number, this reading is the amount the PV falls below the setpoint or reference.) The threshold, deadband, gating, and timing are set in configuration.

Parameters of the P_Alarm object can be accessed by using the following convention: [P_Alarm Name].[P_Alarm Parameter].

Simulation

This object does not have Simulation capability.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	The loop outputs are de-energized. The command source is shown as Program Out of Service. All alarms are cleared.
Powerup (prescan, first scan)	Any commands that are received before first scan are discarded. The loop outputs are de-energized. Embedded P_CmdSrc and P_Alarm instructions are handled in accordance with their standard powerup procedures. See the Reference Manuals for the P_CmdSrc and P_Alarm instructions for details.
Postscan (SFC transition)	No SFC postscan logic is provided.

See the Logix 5000 Controllers Add On Instructions Programming Manual, publication [1756-PM010](#), for more information.

Sequencer Object (P_Seq)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The Rockwell Automation Sequencer Object (P_Seq) provides a flexible controller-based step sequencing solution that reduces engineering time by automating common operator procedures. The step-by-step configuration makes it easy to adjust procedures directly from the HMI displays without having to create or modify custom code in the controller. The Sequencer can be employed in simple and complex sequences without costly re-engineering and testing. You add, delete, or modify steps needed to accomplish your sequence's objective.

Functional Description

The 32 Boolean outputs are used to assert commands to devices. The 32 Real outputs are used to set setpoints or references. Each output (Boolean or Real) can be used optionally in each step, and each output is explicitly defined even if it's not used in a step.

The 32 inputs are used to monitor Boolean signals from devices or logic to determine when a desired state or combination of states have been achieved. When the desired state or combination of states has been achieved, this signals the end of the step.

In operation, when a step is executed, the output values are presented at the Sequencer instruction outputs before the first check of the input conditions. In this way, the output values for each step are present for at least one scan of the Sequencer.

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_Seq_4.10.00_AOI.L5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Step User-defined Data Type

To achieve the greatest flexibility concerning step information storage and number of steps in a sequence, a separate user-defined data type (UDT) is supplied to store the step information (P_SeqStep). You create an array of these UDT members to hold the step configurations of the sequence. Array length is from 2...500 steps. The first step of the array is not available because it's used by the Sequencer instruction for other features and bookkeeping.

Operator Prompt

The P_Prompt instruction can be used with the Sequencer to perform manual prompt operations, such as operator messaging, entering values, or decision-making in the flow of steps.

IMPORTANT See [Operator Prompt \(P_Prompt\) on page 389](#) for the P_Prompt instruction.

For more information, see the following resource:

- Rockwell Automation Sequencer Object Reference Manual, publication [PROCES-RM006](#)

Operations

This section describes the primary operations for this Add-On Instruction.

IMPORTANT See [Appendix A](#) for Command Source and Simulation types.

Alarms

Alarm Name	P_Alarm Name	P_Gate Name	Description
Interlock Trip	IntlkTrip	None	<p>Raised when an interlock 'not OK' condition causes the sequence to perform its configured interlock action. The sequence can:</p> <ul style="list-style-type: none"> • Command the sequence to Stop • Hold at the current step • Transfer control back to the last step configured as an Interlock Fallback Step <p>If interlocks are not bypassed, a bypassable interlock or a non-bypassable interlock 'not OK' condition initiates an interlock trip. If interlocks are bypassed, only a non-bypassable interlock 'not OK' condition initiates an interlock trip.</p>
Sequence Timeout	SeqTO	None	Raised when the amount of time in the current sequence exceeds the step timeout configuration for that step (Cfg_SeqTO).
Step Timeout	StepTO	None	Raised when the amount of time in the current step of the sequence exceeds the step timeout configuration for that step (Ref_Steps[stepnumber].Cfg_FaultT).

Parameters of the P_Alarm object can be accessed by using the following convention: [P_Alarm Name].[P_Alarm Parameter].

Simulation

The P_Seq Add-On Instruction does not have a Simulation capability.

Execution

This table explains the handling of instruction execution conditions.

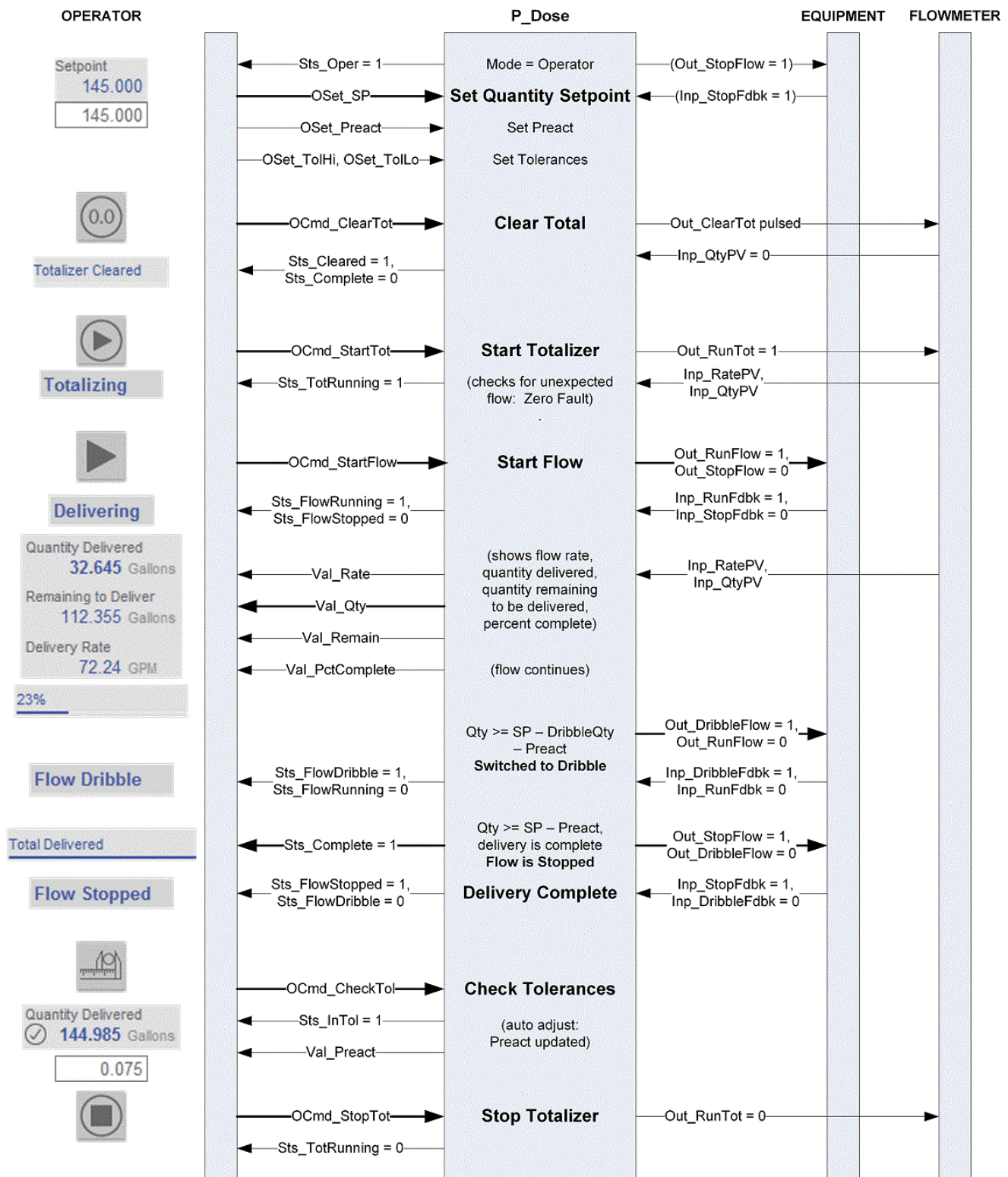
Condition	Description
EnableIn False (false rung)	Command source is set to Program Out of Service. Outputs are left in their last state, and are not being written. Received commands are ignored and cleared.
Powerup (prescan, first scan)	Sequencer is flagged to initialize on first scan. On first scan, the sequence is set to the Idle state and the edit pointer (for online sequence editing HMI displays) is set to step 1 of the sequence.
Postscan (SFC Transition)	No SFC postscan logic is provided.

See the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#), for more information.

Dosing (P_Dose)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The diagram depicts a typical sequence of operator commands, settings, and subsequent instruction actions. The example shows the P_Dose Instruction is performing ingredient dosing in Operator command source.



Functional Description

The primary operations of the P_Dose instruction include the following:

- Provides inputs for rate (flow rate or quantity per time) and quantity (weight, total, or pulse count).
- Provides the ability to use a pulse count as the Quantity PV, with configurable rollover count.
- Provides the ability to totalize the flow rate PV to determine the quantity that is delivered when the flowmeter provides a rate signal but no quantity.
- Provides the ability to calculate the flow rate given the quantity by differentiating with respect to time when the meter or scale provides a total or pulse count but no rate. If the rate PV is calculated from an input quantity, the P_Dose instruction uses a first-order (lag) filter on the calculated rate PV signal to reduce the impact of jitter, scan time, quantization error, or input signal noise.
- Provides linear scaling of the input weight, flow, or quantity value from raw (input card) units to engineering (display) units.
- Provides the ability to clear the totalizer to a zero-quantity starting point for transfer. Once the totalizer is cleared, the instruction checks for unrequested flow, that is, an increase or decrease in total before flow is started, and raises a tare fault alarm if such a weight change occurs.
- Provides a low rate cutoff function, used to ignore flow rate values near zero to deal with noise or zero calibration error in the rate signal.
- Provides the ability to use a flowmeter with built-in totalizer. Forwards the totalizer clear command to the flowmeter and checks that the total was reset. Once the total is cleared, the instruction checks for unrequested flow, an increase in the delivered total before flow is started, and raises a zero fault if such flow occurs.
- Provides outputs to control associated equipment (pumps, valves, and so forth) to start and stop flow. The operator or the program could start the ingredient addition, then pause, and resume it if needed.
- Monitors the status of controlled equipment (pumps, valves, and so forth). Flow is stopped and an alarm is raised on an equipment fault or if the equipment fails to respond as commanded.
- Monitors rate and/or quantity input communication status and provides indication of uncertain or bad rate PV or quantity PV. Flow is stopped and an alarm is raised on a bad PV or communication loss.
- Provides program, operator, or external entry of a quantity to deliver (setpoint) and calculates the quantity remaining to deliver and percent complete during delivery.
- Provides program, operator, or external entry of high and low tolerance limits. Lets the program, operator, or external command source initiate a tolerance check after delivery is complete. Provides a warning if under tolerance and lets the operator bump the flow to make up the shortage. The bump can be configured as a timed bump or as an operator jog-like function. Provides an alarm if over tolerance and inhibits further flow.

- Includes the ability to switch to a lower dribble flow rate automatically as the quantity delivered approaches setpoint. Provides operator or program entry of the dribble quantity. Provides run, dribble, and stop outputs to controlled equipment.
- Uses a preact value to stop flow to account for material in the pipe, time for equipment to stop, and delays in measurement, scan, communication, and so forth. Provides operator, program, or external entry of the preact value. Provides an optional automatic preact correction that is based on the error in delivery when tolerance is checked. The auto correction lets the preact 'learn' the correct value over time.

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_Dose_4.10.00_AOI.L5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Operations

This section describes the primary operations for Add-On Instructions.

IMPORTANT See [Appendix A](#) for Command Source and Simulation types.

Alarms

This instruction uses the following alarms, which are implemented by using embedded P_Alarm and P_Gate Add-On Instructions.

Alarm Name	P_Alarm Name	P_Gate Name	Description
Equipment Fault	EqpFault	None	Raised when the Inp_CtrldEqpFault input is true, or when equipment feedback signals fail to track the commanded state of the equipment within the configured time. If an equipment fault is configured as a shed fault, the flow is stopped and a reset is required to resume flow.

Alarm Name	P_Alarm Name	P_Gate Name	Description
Over Tolerance	OverTol	None	Raised when the tolerance check is performed and the quantity that is delivered exceeds the setpoint by more than the High Tolerance threshold.
Under Tolerance (warning)	UnderTol	None	Raised when the tolerance check is performed and the quantity that is delivered falls short of the setpoint by more than the Low Tolerance threshold. TIP: In some instances, the Bump function can be used to make up the shortage.
Zero fault	ZeroFault	None	Raised if the totalizer fails to clear, or if the totalizer is cleared but then registers flow before flow is commanded to start.

Parameters of the P_Alarm object can be accessed by using the following convention: [P_Alarm Name].[P_Alarm Parameter].

Simulation

When the P_Dose block is placed in simulation, it continues to generate its outputs to control equipment, but the flow quantity and rate inputs are ignored and a simulated flow rate is used.


This operation in simulation is different from the equipment (motor, valve, drive) instructions.



WARNING: In order to avoid starting equipment during simulation, the downstream equipment instructions must be put in simulation so that their outputs are held in the SAFE state.

Failure to do this can result in personal injury or equipment damage.

Set the Inp_Sim parameter to '1' to enable simulation.

Set the Inp_Sim parameter in the controller to '1' to enable simulation. The Simulation icon  is displayed at the top left of the Operator faceplate, and indicates that the device is in simulation.

While in simulation, you can use the following parameters to control how the flow is simulated:

- Cfg_SimRate – the full rate to be used for delivery (in flow units/rate time)
- Cfg_SimDribbleRate – the rate to be used for dribble (in flow units/rate time)

When you have finished simulation, clear the Inp_Sim parameter to 0 to return to normal operation.

Execution

The following table explains the handling of instruction execution conditions.

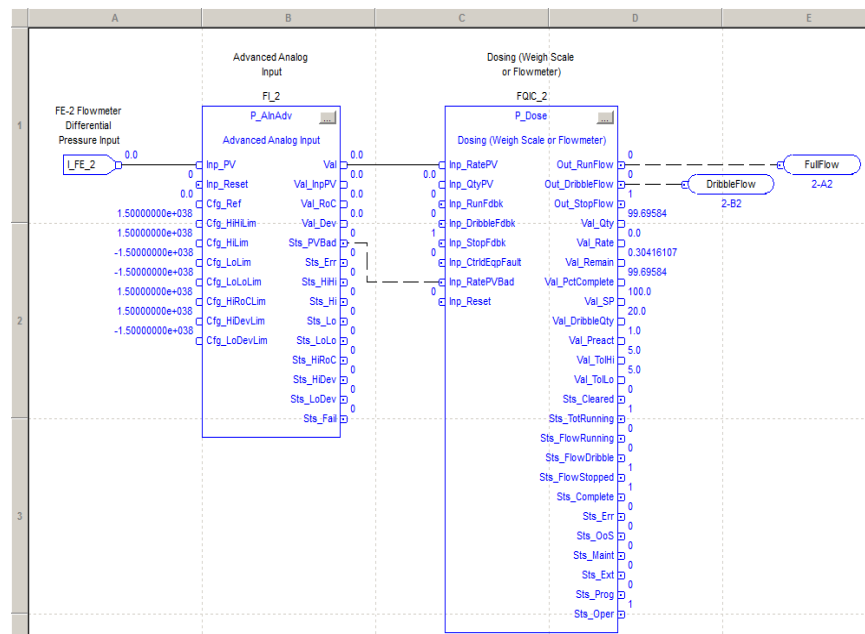
Condition	Description
EnableIn False (false rung)	Any commands that are received are discarded. All alarms are cleared. The command source is reported as Program Out of Service. The displayed rate is zeroed. Outputs to controlled equipment are de-energized. Other output parameters (values and status) hold their last value.
Powerup (prescan, first scan)	Any commands that are received before first scan are discarded. Embedded P_Alarm instructions are handled in accordance with their standard powerup procedures. See the reference manual for the P_Alarm instructions for more information.
Postscan (SFC transition)	No SFC postscan logic is provided.

See the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#), for more information.

Programming Example

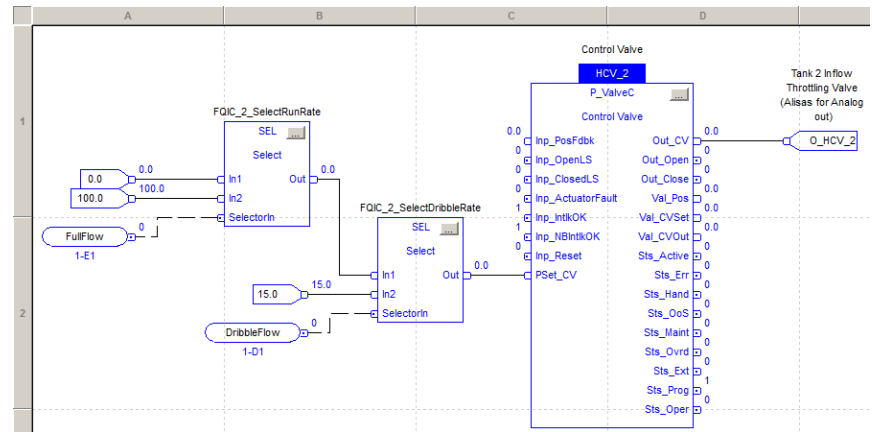
This example uses a flow measuring element, such as an orifice plate, that provides a differential pressure (DP) based on flow. A basic differential pressure transmitter provides the DP signal to the controller.

- The P_InAdv instruction is configured to convert the DP signal to a flow, using the advanced analog input's square root characterization feature (Cfg_UseSqRt = 1).
- The flow signal is tied to the Inp_RatePV input of the P_Dose block. The dosing block is configured to integrate the flow rate signal to accumulate the quantity transferred (Cfg_CalcQty = 1).
- Configure the analog input to filter the signal slightly (Cfg_FiltTC = 0.1 sec) to account for signal noise.
- The dosing block low flow cutoff is set to 0.05 GPM (Cfg_LoRateCutoff = 0.05) to alleviate any calibration anomalies.



The dosing block connects to a P_ValveC instruction to control an analog control valve.

- When full flow is selected, the SEL blocks select a value of 100.0 to go to the P_ValveC Program setting for the output to the valve (PSet_CV) and the valve is wide open.
- When dribble flow is selected, the SEL block select a value of 15.0 to go to the valve to throttle down to 15% open.
- When the dosing block selected to stop flow, both the DribbleFlow and FullFlow signals are 0, and the SEL blocks default to a CV of 0.0 to close the valve.



Lead/Lag/Standby Motor Group (P_LLS)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_LLS (Lead Lag Standby motor group) Add-On Instruction provides control of a parallel group of motors. Such groups are commonly used for a group of pumps that maintain pressure on a header despite wide changes in demand, such as in municipal-scale or plant-scale water distribution.

Functional Description

The P_LLS Instruction controls and monitors a group of 2 to 30 motors and provides:

- Operator, Program, and Override capability to start and stop the group (as a group).
- Ability for the Operator or Program to enter a 'demand', the number of motors to run.
- Configurable maximum demand (1 to number of motors in group).
- Configurable minimum demand (0 to maximum demand).
- Configurable to stop the last started motor or the first started motor (first-on-last-off or last-on-last-off).
- Configurable delay between starts and configurable delay between stops.
- Start and Stop commands on the P_LLS instruction allow for starting or stopping the motors as a group. The delay between starts or stops can be configured to sequence the motors.
- Starts or stops motors as required to meet the entered demand.
- Identifies (and optionally alarms) when there are not enough motors available to start (in Program Mode and ready to run) for the given demand to be met.
- Identifies (and optionally alarms) when there are not enough motors available to stop (in Program Mode and ready to stop) for the given demand to be met.
- Ability to rotate the list of motors (demote the lead, promote the others).
- Monitoring of Permissive conditions to allow starting the motor group.
- Monitoring of Interlock conditions to stop/prevent starting the motor group.
- Alarm if interlock conditions cause the group to be stopped.
- Supports HMI 'breadcrumbs' for Alarm Inhibited, Bad Configuration, Not Ready, and Maintenance Bypass Active.
- 'Available' status for use by automation logic to know whether motor group can be controlled by other objects.

This instruction controls a group of motors, such as a set of pumps with common intake source and discharge destination. The number of motors to run depends on the demand on the system. The P_LLS group can be configured to consist of 2...30 motors.

The minimum demand can be set as low as 0, so that all motors are stopped at minimum demand. The maximum demand can be set as high as the number of pumps in the group. (In this case, if the demand were as high as the number of pumps in the group, there would be no 'standby' pumps.)

The P_LLS instruction uses a sorting algorithm to deal with motors that are not available. If a motor is running and not available (perhaps running in Operator command source), the motor is forced to the top of the sort. If a motor is stopped and not available (faulted), the motor is forced to the bottom of the sort. The motors that are available to start and stop are controlled to meet the demand. If the demand cannot be met because of unavailable motors, a status/alarm is provided.

EXAMPLE Two motors in a group of four are stopped and not available. The P_LLS instruction raises a 'can't start' alarm when the demand reaches three because there are only two motors available to run.

The P_LLS instruction uses an array of structures of the type 'P_LLS_Motor' to interface to the motors. Each interface element in the array provides the signals that are required between the P_LLS instruction and one motor.

Configuration data for the motor are also provided in the array. This data includes Priority and Preference values that can be used to affect the sorting of the motors.

A Maintenance 'out of service' flag that removes a motor from consideration in the sort is also included. The interface also includes a 'user sort' value that can be used, for example, to push motors up or down the sort based on accumulated runtime or other criteria.

InOut parameters are used to link the Add-On Instruction to external tags that contain necessary data for the instruction to operate. These external tags must be of the data type shown.

Table 9 - P_PF52x Drive InOut Parameters

Tag Name	Data Type	Description
Ref_Motors	P_LLS_Motor []	Motor interface array (link of 2...32 motors)

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_LLS_4.10.00_AOIL5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

In the following example, the interface tag is named 'MyP_LLS_Motors'. The type is an array of P_LLS_Motor structures. For each group of motors, create an interface array. Name its tag the same as the P_LLS instruction backing tag, plus '_Motors'. The array must have at least as many elements as there are motors in the group.

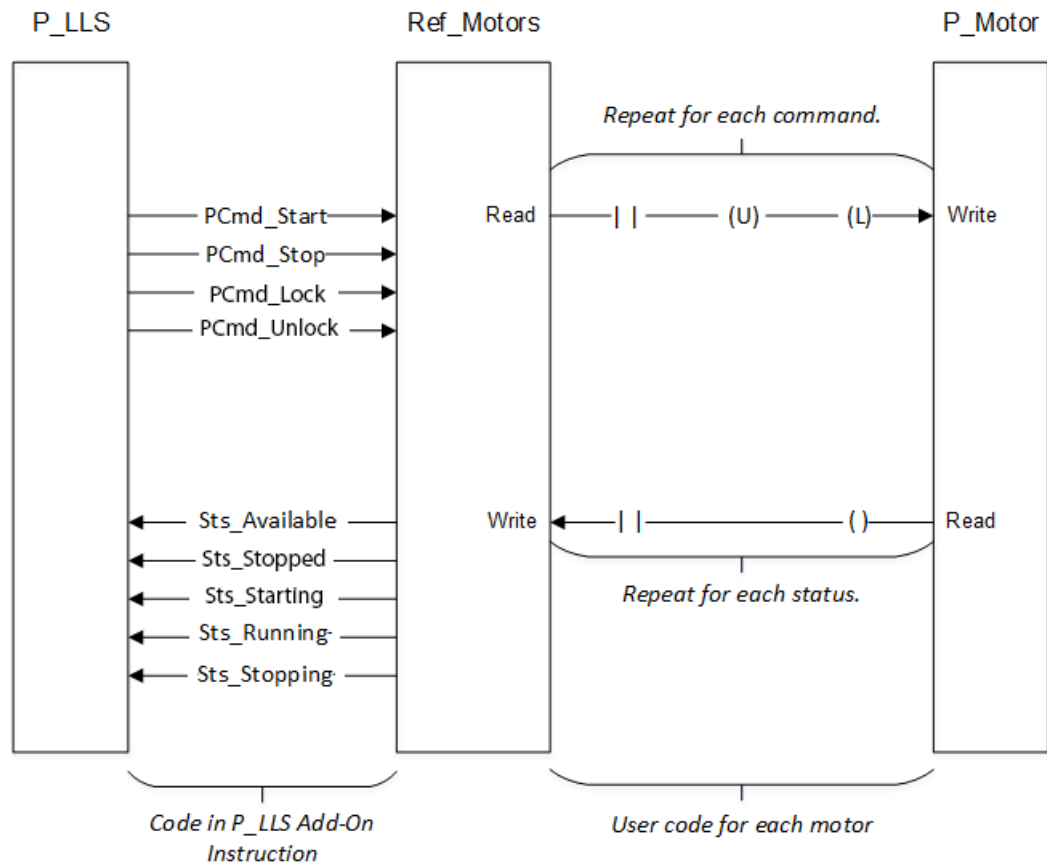
Scope:	ProcessObjects_4	Show:	All Tags	▼	Enter Name Filter...
Name	Value	Force Mask	Style	Data Type	Description
- MyP_LLS_Motors	{ ... }	{ ... }		P_LLS_Motor[4]	Interfaces to P_LLSGroup Group's Motors
+ MyP_LLS_Motors[0]	{ ... }	{ ... }		P_LLS_Motor	Interfaces to P_LLSGroup Group's Motors One motor interface row (for using in P_LL
+ MyP_LLS_Motors[1]	{ ... }	{ ... }		P_LLS_Motor	Interfaces to P_LLSGroup Group's Motors One motor interface row (for using in P_LL
+ MyP_LLS_Motors[2]	{ ... }	{ ... }		P_LLS_Motor	Interfaces to P_LLSGroup Group's Motors One motor interface row (for using in P_LL
+ MyP_LLS_Motors[3]	{ ... }	{ ... }		P_LLS_Motor	Interfaces to P_LLSGroup Group's Motors One motor interface row (for using in P_LL

This tag is an array of parameter values that facilitates communications between P_LLS and an instance of P_Motor. The following table shows the contents of each member of the array.

Table 10 - Array Member Content

Name	Data Type	Description
Inp_OtherSel	DINT	Other motor selection criteria (0...255) (input to LLS).
Inp_Demote	BOOL	Demote this motor to bottom of list (for example, on high runtime) (input to LLS).
Cfg_Prio	DINT	Motor priority in list (0...31 -- if unused, set to 0).
Cfg_Pref	DINT	Motor preference in list (0...31), all else being equal.
Cfg_NavTag	STRING_NavTag	Logix Tag to navigate to for this motor (For example, P_Motor backing tag name).
PCmd_Start	BOOL	Program Command to start motor (output from LLS).
PCmd_Stop	BOOL	Program Command to stop motor (output from LLS).
PCmd_Lock	BOOL	Command to Acquire and Lock motor in Program (output from LLS).
PCmd_Unlock	BOOL	Command to Unlock motor from Program (output from LLS).
Sts_Available	BOOL	Motor is in Program command source and ready to operate (input to LLS).
Sts_Stopped	BOOL	Motor is confirmed stopped (input to LLS).
Sts_Starting	BOOL	Motor is starting (input to LLS).
Sts_Running	BOOL	Motor is confirmed running (input to LLS).
Sts_Stopping	BOOL	Motor is stopping (input to LLS).
Val_Rank	DINT	This motor's current rank in the list (1=Lead, 2=Lag, ...)

The following image shows the relationship between P_LLS, Ref_Motors (interface), and P_Motor.



The following images show an example of the ladder logic for transferring commands and motor status for one motor. Three steps are shown:

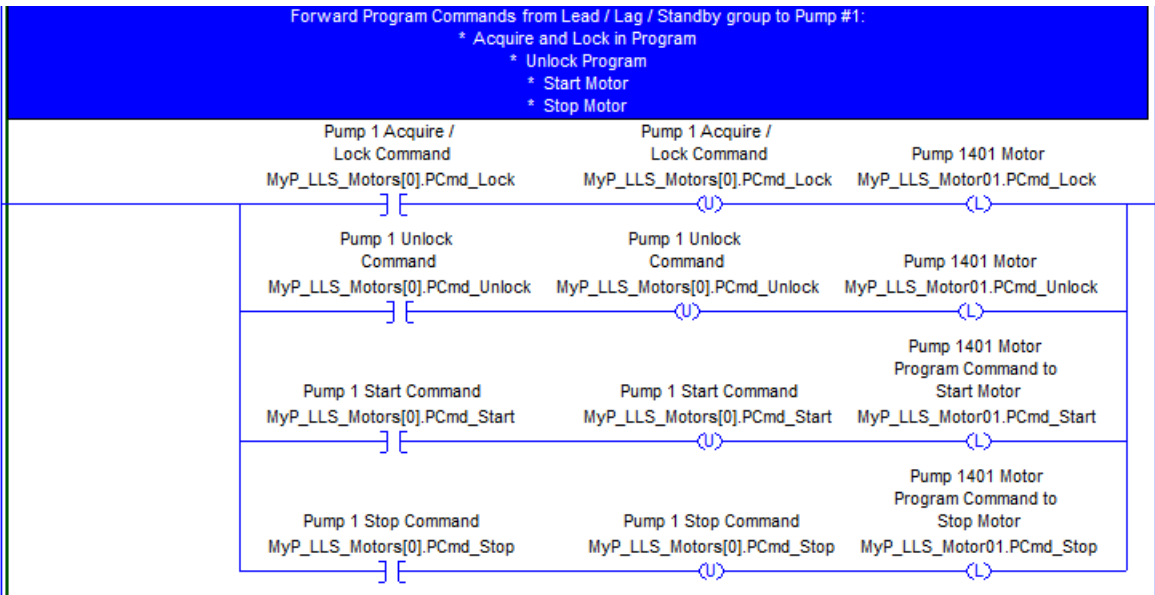
- Forward commands to the motor from P_LLS.
- Execute the motor logic.
- Return motor status back to P_LLS.

Each of the three steps is shown on its own rung, if desired, all three steps can be in one branched rung.

In the following diagram, the process for forwarding each of the commands (PCmd_Acq, PCmd_Rel, PCmd_Start, and PCmd_Stop) is:

- The appropriate bit in the interface is tested to see if it set.

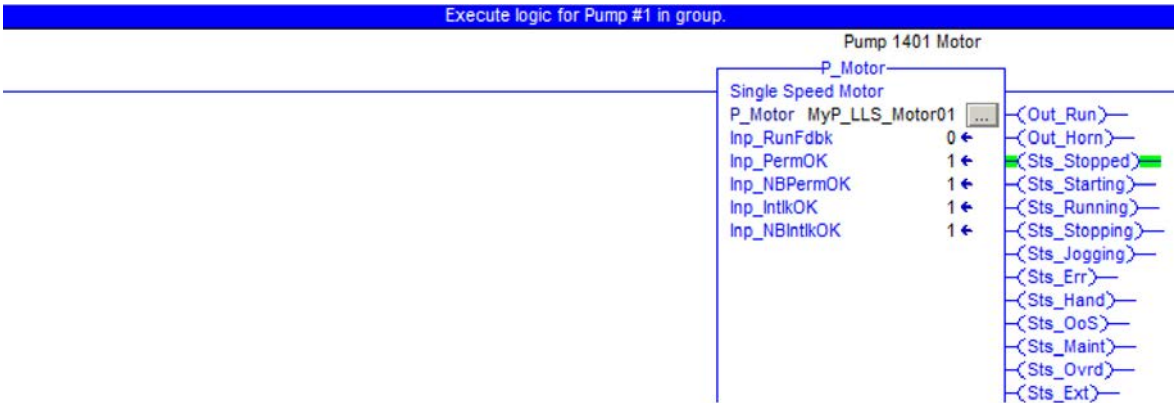
If the bit is set, the bit is cleared and the corresponding program command on the motor is set.



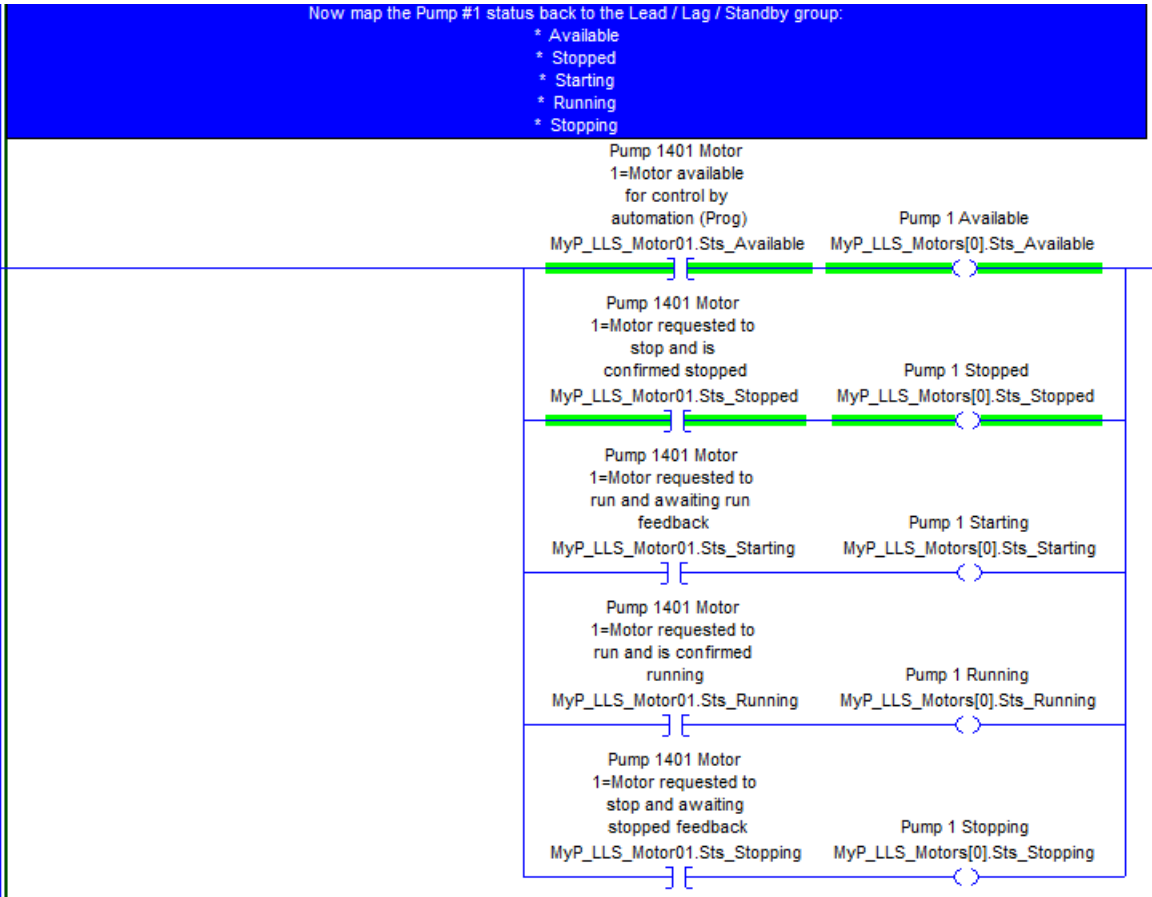
TIP The interface for the first motor in the group is element [0] of the interface array tag.

Next, the motor logic is executed.

The motor logic uses the program commands to control the physical motor. The motor logic also receives feedback from the motor.



The status (available, stopped, starting, running, and stopping) is read from the motor and written to the interface.



Operations

This section describes the primary operations for Add-On Instructions.

Alarms

This instruction uses the following alarms, which are implemented by using embedded P_Alarm and P_Gate Add-On Instructions.

Alarm Name	P_Alarm Name	P_Gate Name	Description
Can't Start	CantStart	None	Raised when there are not enough motors available to start to satisfy the entered Demand. Too many motors are faulted or stopped in a command source other than Program.
Can't Stop	CantStop	None	Raised when there are not enough motors available to stop to satisfy the entered Demand. Too many motors are running in a command source other than Program.
Interlock Trip	IntlkTrip	None	Raised when the motor group is running and an interlock 'not OK' condition causes the group to stop. If interlocks are not bypassed, a bypassable interlock or a non-bypassable interlock 'not OK' condition initiates an interlock trip. If interlocks are bypassed, only a non-bypassable interlock 'not OK' condition initiates an interlock trip.

Parameters of the P_Alarm object can be accessed by using the following convention: [P_Alarm Name].[P_Alarm Parameter].

Simulation

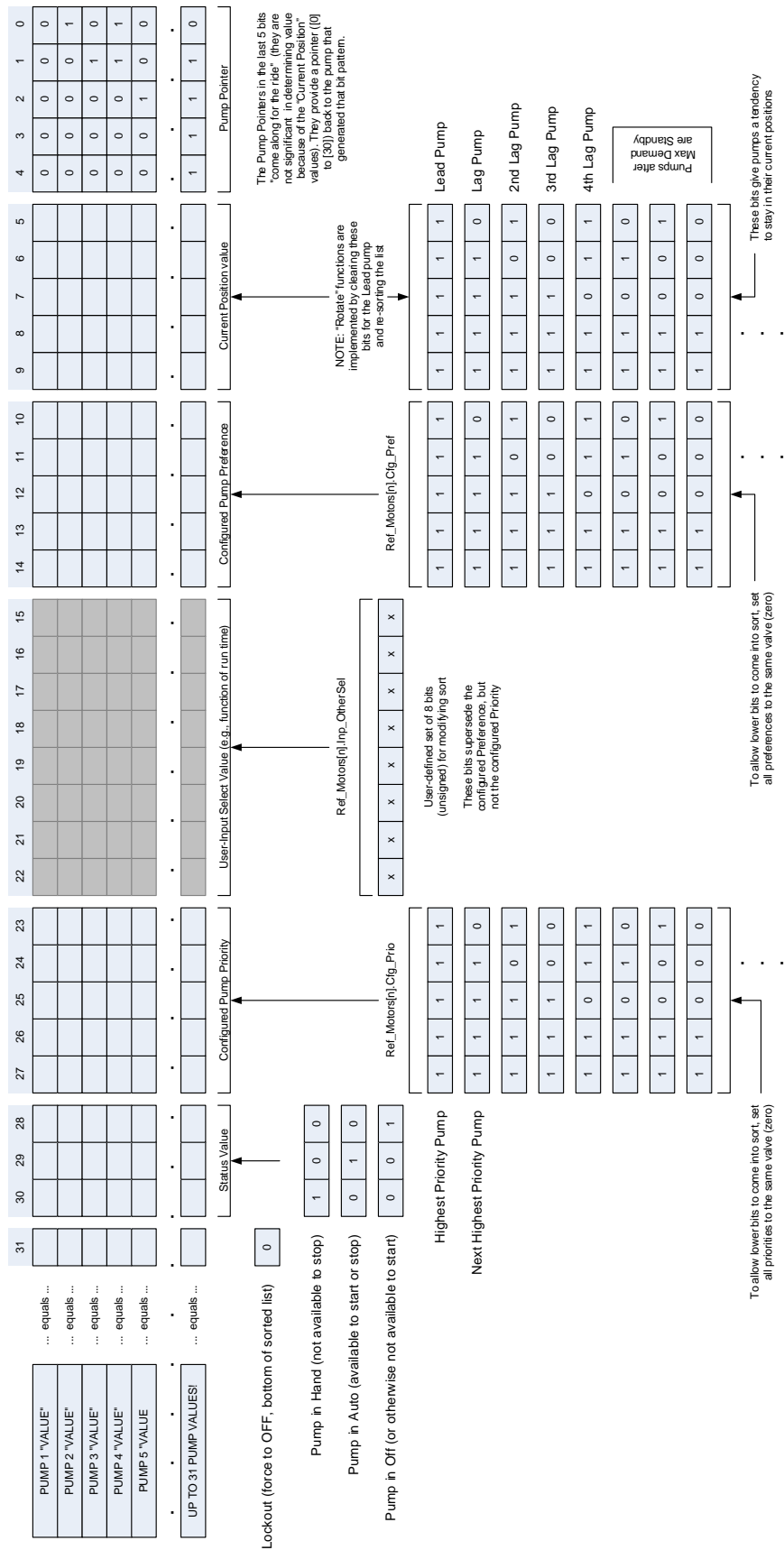
P_LLS does not have a simulation capability.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	Processing for EnableIn False (false rung) is handled the same as if the group were taken out of service by Command. The group outputs are de-energized and the group is shown as Program Out of Service on the HMI. All alarms are cleared.
Powerup (prescan, first scan)	Any commands received before first scan are discarded. The motor is de-energized and treated as if it were commanded to stop. Embedded P_CmdSrc and P_Alarm instructions are handled in accordance with their standard powerup procedures. See the Reference Manuals for the P_CmdSrc and P_Alarm instructions for details.
Postscan (SFC transition)	No SFC postscan logic is provided.

See the, Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#), for more information.



KEY CONCEPT: The list of pumps is sorted (by descending numeric value, based on signed integer bit pattern) to determine Lead / Lag / 2nd Lag ... order.

Motor Sort Algorithm

To determine the order in which the motors (pumps) are started, signed integer bit patterns for each motor are sorted by numeric value. The following list is the order in which the bit patterns are evaluated when sorting:

- Out-of-service bit
- Status value
- Priority value
- User-input value
- Preference value
- Current position value

Out of Service (Bit 31)

This bit is used to flag the motor out of service (value = 1) and automatically send it to the bottom of the list. If this bit = 0, the motor is free to operate and bits 5...30 determine its start order.

If more than one motor is out of service, bits 5...30 determine their position at the bottom of the list.

Out-of-service motors are not commanded and are not counted as running even if actually running.

Status Value (Bits 30...28)

The status of the motor determines the value of these bits:

- 100 - The motor is in Hand and is not available to stop
- 010 - The motor is in Auto and is free to start or stop
- 001 - The motor is Off and is not available to start

If all motors have the same value, these bits do not affect the sort; the next set of bits becomes the determining factor in the sort.

Priority Value (Bits 27...23)

These bits are next in the order of precedence for sorting the array list. The value of these bits corresponds to the number entered in the Motor Priority field in the Motor Configuration dialog box.

The highest priority value has a pattern of '11111' (31), the next highest priority value is '11110' (30), and so forth.

If this priority is not to be used for the sort, set the priority value to zero for every motor.

If all motors have the same value, these bits do not affect the sort; the next set of bits becomes the determining factor in the sort.

User-input Values (Bits 22...15)

If the Status Values are equal and the Priority values are equal, enter values in these bits to sort the motors in the array list to the desired order.

The highest user-input value has a pattern of '11111111' (255), the next highest user-input value is '11111110' (254), and so forth.

If this value is not to be used for the sort, set the value to zero for every motor.

If all motors have the same value, these bits do not affect the sort; the next set of bits becomes the determining factor in the sort.

Preference Value (Bits 14...10)

These bits are next in the order of precedence for determining the order of the motors in the array list. The value of these bits corresponds to the number entered in the Motor Preference field in the Motor Configuration dialog box.

The highest preference value has a pattern of '11111' (31), the next preference value is '11110' (30), and so forth.

If this value is not to be used for the sort, set the value to zero for every motor.

If all motors have the same value, these bits do not affect the sort; the next set of bits becomes the determining factor in the sort.

Current Position (Bits 9...5)

IMPORTANT The current position bits are the only set of bits that cannot be equal.

These bits are next in the order of precedence for determining the order of the motors in the array list. The value of these bits corresponds to the value of the current position of the motor in the list, and the value is established by the P_LLS instruction. There is no user entry for this field.

- Lead motor - '11111' (31)
- First Lag motor - '11110' (30)
- Second Lag motor - '11101' (29) and so on ...

The Status Value Priority value, User-input value, and Preference Value must be equal for all motors for the Current Position to be a determining factor in the sort.

Motors

Purpose

This chapter is for the operation of the Add-on Instructions. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The Process Objects in this group provide control and monitoring for smart motor controllers, drives, and overload relays. [Table 10](#) describes the objects in this group, including when to use and not to use each one.

Table 10 - Motors

Process Object	Description	When to Use	When Not to Use
Single-speed Motor (P_Motor)	<p>This instruction controls a non-reversing, single-speed motor and monitors for fault conditions.</p> <p>The motor can use a full voltage starter (FVNR), a soft starter, or other motor protective equipment, and can optionally provide run feedback. The P_Motor instruction includes faceplates and graphic symbols for operator display and manipulation. The instruction provides alarms for several fault conditions.</p> <p>The instruction also provides run feedback and a display of actual motor status.</p>	<ul style="list-style-type: none"> • Want to control a single-speed (running or stopped) motor. • Motor can use a full voltage starter (FVNR), a soft starter, or other motor protective equipment. • Can optionally provide run feedback. 	<ul style="list-style-type: none"> • Want to control a two-speed (fast/slow/stopped) motor. Use the P_Motor2Spd Two-speed Motor instruction instead. • Want to control a reversing (forward/stopped/reverse) motor. Use the P_MotorRev Reversing Motor instruction instead. • Want to control a motor with continuously variable speed. Use the P_VSD Variable-speed Drive instruction instead. • Want to control a motor that is part of a valve actuator. Use the P_ValveMO Motor-operated Valve instruction instead. • Monitor, and optionally trip, a locally operated (hand-operated) motor. The motor can be single-speed, two speed, or reversing. Use the P_MotorHO Hand-operated Motor instruction instead.

Table 10 - Motors

Process Object	Description	When to Use	When Not to Use
Two-speed Motor (P_Motor2Spd)	<p>This instruction controls a non-reversing, two-speed motor (fast/slow/stopped) and monitors for fault conditions. The motor can optionally have run feedback that, if available, is used to confirm that the motor is running at the commanded speed, and alarm if not.</p> <p>The instruction also does the following:</p> <ul style="list-style-type: none"> • Detects failure of the motor to start or stop and generates appropriate alarms. • Provides for simulation of a working motor that disables outputs, for use in off-process training, testing, or simulation. 	<ul style="list-style-type: none"> • Use this instruction when you operate a two-speed motor that runs in one direction. • Supports motors that have three controlled states: running fast, running slow, and stopped. • Motor can optionally have run feedback that is used to confirm that the motor is running at the commanded speed. Prompts an alarm if not at the commanded speed. 	<ul style="list-style-type: none"> • Operating a single-speed motor (running/stopped only). Use the P_Motor Single-Speed Motor instruction instead. • Operating a continuously variable speed motor, such as one wired to a variable-frequency AC drive or variable-speed DC drive. Use the P_VSD Variable Speed Drive instruction instead. • Operating a simple reversing motor (forward, reverse, and stopped only). Use the P_MotorRev Reversing Motor instruction instead. • Monitoring, and optionally tripping, a locally operated (hand-operated) motor. The motor can be single-speed, two speed, or reversing. Use the P_MotorHO Hand-operated Motor instruction instead.
Reversing Motor (P_MotorRev)	<p>This instruction controls a reversing motor (forward/reverse/stopped) and monitors for fault conditions. The motor can optionally have run feedback, that, if available, is used to confirm that the motor is running in the commanded direction, and alarm if not.</p> <p>The instruction also does the following:</p> <ul style="list-style-type: none"> • Monitors permissive and interlock conditions • Detects failure of the motor to start or stop and generates appropriate alarms. 	<ul style="list-style-type: none"> • Use this instruction when you operate a simple reversing motor. • Supports motors that have three controlled states: running forward, running reverse, and stopped. • Motor can optionally have run feedback that is used to confirm that the motor is running in the commanded direction. Prompts an alarm if not at the commanded direction. 	<ul style="list-style-type: none"> • Operating a single-speed non-reversing motor (running/stopped only). Use the P_Motor instruction instead. • Operating a continuously variable speed motor, such as one wired to a variable-frequency AC drive or variable-speed DC drive. Use the P_VSD instruction instead. • Operating a two-speed motor that runs in one direction (fast, slow, stopped only). Use the P_Motor2Spd instruction instead. • Monitoring, and optionally tripping, a locally operated (hand-operated) motor. The motor can be single-speed, two speed, or reversing. Use the P_MotorHO instruction instead.
Hand-operated Motor (P_MotorHO)	<p>This instruction monitors a locally controlled (hand-operated) motor. The instruction supports single-speed motors (running or stopped), two-speed motors (running fast, running slow, or stopped), and reversing motors (running forward, running reverse, and stopped). The motor must provide run feedback. The instruction also supports an optional trip function and output that is used to stop the motor.</p>	<ul style="list-style-type: none"> • The optional trip function provides the following capabilities: <ul style="list-style-type: none"> – Detects failure to stop when tripped and generate an appropriate alarm. – Monitors interlock conditions to trip the motor, and alarm when an interlock stops a running motor. – Provides for simulation of a working motor while disabling the trip output, for use in off-process training, testing, or simulation. – Monitors I/O communication, and alarm (and trip if the Shed On I/O Fault function is enabled) on a communication fault. 	<ul style="list-style-type: none"> • Need to do more than monitor or trip the motor. • Need to operate a single-speed motor (running/stopped). Use the P_Motor Single-Speed Motor instruction instead. • Need to operate a continuously variable speed motor, such as one wired to a variable-frequency AC drive or variable-speed DC drive. Use the P_VSD Variable Speed Drive instruction instead. • Need to operate a two-speed motor that runs in one direction (fast, slow, or stopped). Use the P_Motor2Spd Two Speed Motor instruction instead. • Need to operate a simple reversing motor (forward/reverse/stopped). Use the P_MotorRev Reversing Motor instruction instead.

Table 10 - Motors

Process Object	Description	When to Use	When Not to Use
PowerFlex 523/525 Drives (P_PF52x)	<p>This instruction is used to control and monitor a PowerFlex® 523 or PowerFlex 525 variable-frequency drive with optional EtherNet/IP Interface.</p> <p>The P_PF52x instruction includes the following capabilities:</p> <ul style="list-style-type: none"> Starting, stopping, jogging of the drive, and setting speed reference and direction Supports HMI 'breadcrumbs' for Alarm Inhibited, Bad Configuration, Not Ready, and Maintenance Bypass Active. <p>This instruction is provided as a rung import for installation.</p>	<ul style="list-style-type: none"> Controls and monitors a PowerFlex 525 variable-frequency drive with embedded or optional add-on EtherNet/IP Interface. Collects and displays diagnostic information from the drive by using data links on the EtherNet/IP interface. 	<ul style="list-style-type: none"> Do not use this instruction with other Allen-Bradley® drive families. There is a dedicated instruction for the PowerFlex 753 variable-frequency drive that uses the 20-COMM-E EtherNet/IP interface (P_PF753). There is also a dedicated instruction for the PowerFlex 753 or 755 variable-frequency drive that uses the built-in or add-on 20-750-series EtherNet/IP interfaces. Using a PowerFlex 6000 or PowerFlex 7000 drive. It is highly recommended that other Allen-Bradley® (or non-Allen-Bradley) drive families use the P_VSD generic Variable Speed Drive instruction.
PowerFlex 753 Drive (P_PF753)	<p>This instruction is used to operate one variable-speed motor using a PowerFlex 753 AC variable-frequency drive and monitoring for fault conditions.</p> <p>This instruction is designed to work with the PowerFlex 753 drive and a 20-COMM-E Ethernet communication module.</p> <p>The P_PF753 instruction includes the following capabilities:</p> <ul style="list-style-type: none"> Provides alarms and drive shutdown for Fail to Start and Fail to Stop if the feedback does not follow the commanded state within a configured amount of time. <p>This instruction is provided as a rung import for installation.</p>	<ul style="list-style-type: none"> Need to operate a motor that is connected to a PowerFlex 753 variable frequency AC drive that is communicating with the controller over an EtherNet/IP network. Need the ability to start and stop the drive and motor, control the drive speed (via speed reference), and monitor the drive run status and speed feedback to verify that the drive is running or stopped. Maintenance personnel can disable (soft lock out) the drive. This capability is not a substitute for hard lockout/tagout (LOTO) procedures. 	<ul style="list-style-type: none"> Need to operate a single-speed motor (running and stopped only). Use the P_Motor instruction instead. Need to operate a two-speed motor (fast, slow, and stopped only). Use the P_Motor2Spd instruction instead. Need to operate a simple reversing motor (forward, reverse, and stopped only). Use the P_MotorRev instruction instead. Need to operate a motor with multiple discrete speeds. You need specific logic for this motor. The P_PF753 instruction is designed for motors with continuously variable (analog) speed, not multiple discrete speed selections. You can use the P_D4SD or P_nPos instruction for motors with multiple discrete speeds. If you are operating a PowerFlex 753 with an enhanced Ethernet card (20-750-series EtherNet/IP.) In this case use the P_PF755 instruction.

Table 10 - Motors

Process Object	Description	When to Use	When Not to Use
PowerFlex 755, PowerFlex 755TL/TR Drives (P_PF755)	<p>This instruction is used to operate one variable-speed motor using a PowerFlex 755 AC variable-frequency drive and monitoring for fault conditions.</p> <p>This instruction is designed to work with a PowerFlex 755, PowerFlex 755TL, or PowerFlex 755TR variable frequency AC drive that is communicating with the controller over an EtherNet/IP network. The instruction also works with a PowerFlex 753 drive with an enhanced Ethernet card.</p> <p>This instruction is provided as a rung import for installation.</p>	<ul style="list-style-type: none"> • Need the ability to start and stop the drive and motor, control the drive speed (via speed reference), and monitor the drive run status and speed feedback to verify whether the drive is running or stopped. • Need alarms and drive shut down for Fail to Start and Fail to Stop if the feedback does not follow the commanded state within a configured amount of time. • Need the ability to read a fault code from the drive and provide descriptive text of fault codes. • Need an optional capability to support reversing drives, with commands for forward and reverse rotation, and display of actual rotation direction. 	<ul style="list-style-type: none"> • Operating a single-speed motor (running and stopped only). Use the P_Motor instruction instead. • Operating a two-speed motor (fast, slow, and stopped only). Use the P_Motor2Spd instruction instead. • Operating a simple reversing motor (forward, reverse, and stopped only). Use the P_MotorRev instruction instead. • Operating a motor with multiple discrete speeds. You need specific logic for this motor. The P_PF755 instruction is designed for motors with continuously variable (analog) speed, not multiple discrete speed selections. You can use the P_D4SD or P_nPos instruction for motors with multiple discrete speeds.
PowerFlex 6000 Drive (P_PF6000)	<p>This instruction operates one variable-speed motor using a PowerFlex 6000 medium voltage variable frequency AC drive. The drive operates via an add-on EtherNet/IP interface.</p> <p>The instruction controls the drive and monitors fault conditions.</p> <p>This instruction is provided as a rung import for installation.</p>	<ul style="list-style-type: none"> • Need ability to start and stop the drive and motor, control the drive speed (via speed reference). Also, monitor the drive run status and speed feedback to verify whether the drive is running or stopped. • Need alarms and drive shut down for Fail to Start and Fail to Stop if the feedback does not follow the commanded state within a configured amount of time. • Need ability to read fault data from the drive and provide descriptive text with fault codes. 	<ul style="list-style-type: none"> • Need to operate a single-speed motor (running and stopped only). Use the P_Motor instruction instead. • Need to operate a two-speed motor (fast, slow, and stopped only). Use the P_Motor2Spd instruction instead. • Need to operate a simple reversing motor (forward, reverse, and stopped only). Use the P_MotorRev instruction instead. • Need to operate a motor with multiple discrete speeds. You need specific logic for this motor. The P_PF6000 instruction is designed for motors with continuously variable (analog) speed, not multiple discrete speed selections. You can use the P_D4SD or P_nPos instruction for motors with multiple discrete speeds.
PowerFlex 7000 Drive (P_PF7000)	<p>This instruction operates one variable-speed motor by using a PowerFlex 7000 medium voltage variable frequency AC drive. The instruction controls the drive in various modes and monitors fault conditions.</p> <p>Instruction is designed for motors with continuously variable (analog) speed, not discrete speed selections.</p> <p>This instruction is provided as a rung import for installation.</p>	<ul style="list-style-type: none"> • Must operate a motor that is connected to a PowerFlex 7000 variable frequency AC drive and communicates with the controller over an EtherNet/IP network. • Instruction is designed to work with the Studio 5000 Logix Designer® application, Version 18 and later. 	<ul style="list-style-type: none"> • Operating a single-speed motor (running and stopped only). Use the P_Motor instruction instead. • Operating a two-speed motor (fast, slow, and stopped only). Use the P_Motor2Spd instruction instead. • Operating a simple reversing motor (forward, reverse, and stopped only). Use the P_MotorRev instruction instead. • Operating a motor with multiple discrete speeds. You need specific logic for this motor. You can use the P_D4SD or P_nPos instruction for motors with multiple discrete speeds.

Table 10 - Motors

Process Object	Description	When to Use	When Not to Use
SMC-50 Smart Motor Controller (P_SMC50)	<p>This instruction controls and monitors a motor via an SMC™-50 Smart Starter.</p> <p>This instruction communicates with the motor controller to start, stop, and jog the motor. The instruction also monitors the status of the motor, detects motor failure to start or stop, and displays motor runtime information. The runtime data includes power, power factor, motor thermal usage, time to trip, time until reset, and motor controller fault codes (with text display of fault cause).</p> <p>This instruction is provided as a rung import for installation.</p>	<ul style="list-style-type: none"> • Use this instruction to monitor and control a motor using an Allen-Bradley® SMC-50 series Smart Motor Controller (soft starter) by using its built-in EtherNet/IP interface. • Other capabilities include: <ul style="list-style-type: none"> – Simulation, providing feedback of a working motor and starter while disabling outputs – Monitoring of I/O communication faults – Alarms for Fail to Start, Fail to Stop, Interlock Trip, Motor/Starter Fault, and I/O Fault – Option to reset faults and alarms automatically when an operator commands a motor to start or stop 	<ul style="list-style-type: none"> • Do not use this instruction with other Smart Motor Controllers, variable speed drives, across-the-line starters, and so on. • For the SMC Flex series of Smart Motor Controllers, use the P_SMCFlex Add-On Instruction instead. • For variable speed drives, use the Add-On Instruction for that drive family (P_PF753 for PowerFlex 753, P_PF755 for PowerFlex 755, and P_PF52x for PowerFlex 523 and PowerFlex 525 families) or the P_VSD generic Variable Speed Drive Add-On Instruction. • For across-the-line starters, use the P_Motor (single-speed motor), P_MotorRev (reversing motor), P_Motor2Spd (two-speed motor), or use the P_D4SD (Discrete 2-, 3-, or 4-State Device) Add-On Instruction.
SMC Flex Smart Motor Controller (P_SMCFlex)	<p>This instruction controls and monitors a motor via an SMC Flex series Smart Motor controller (soft starter).</p> <p>This instruction communicates with the motor controller to start and stop the motor. The instruction also monitors the status of the motor, detects motor failure to start, stop, or motor controller faults, and displays motor runtime information. The runtime data includes phase currents, motor power and power factor, and motor controller fault codes.</p> <p>This instruction is provided as a rung import for installation.</p>	<ul style="list-style-type: none"> • The SMC Flex Smart Motor Controller also provides: <ul style="list-style-type: none"> – Monitoring of Permissive conditions to allow starting – Monitoring of Interlock conditions to stop/prevent starting – Simulation, which provides feedback of a working motor and starter while disabling outputs – Monitoring of I/O communication faults – Option to reset faults and alarms automatically when operator commands motor to start or stop 	<ul style="list-style-type: none"> • Do not use this instruction with other Smart Motor Controllers, variable speed drives, across-the-line starters, and so forth. <ul style="list-style-type: none"> – For the SMC 50 series of Smart Motor Controllers, use the P_SMC50 Add-On Instruction instead. – For variable speed drives, use the Add-On Instruction for that drive family: (P_PF753 for PowerFlex 753, P_PF755 for PowerFlex 755, P_PF52x for PowerFlex 523 and PowerFlex 525 families, P_VSD Add-On Instruction. – For across-the-line starters, use P_Motor (single speed motor), P_Motor (single speed motor), P_MotorRev (reversing motor), P_Motor2Spd (two-speed motor) Add-On Instruction. For three-speed motors, use the P_D4SD (discrete 2-, 3-, or 4-state device) Add-On Instruction.

Table 10 - Motors

Process Object	Description	When to Use	When Not to Use
Variable-speed Drive (P_VSD)	<p>This instruction operates one variable speed motor by using a drive (AC variable frequency or DC) in various modes, monitoring for fault conditions.</p> <p>This instruction is designed to work with all currently available and many legacy Allen-Bradley® drives, including:</p> <ul style="list-style-type: none"> Bulletin 1336 PLUS™ II Bulletin 1395 PowerFlex 4/40/70/700 PowerFlex DC Drives <p>This instruction also works with drives and other variable-speed motor control products via digital I/O for the start/stop/running signals. You can also use analog I/O for the speed reference and speed feedback signals.</p>	<p>Use the instruction for the following:</p> <ul style="list-style-type: none"> Control of the drive through the standard P_CmdSrc Add-On Instruction. Scaling of the speed reference from user (engineering) units, such as RPM, to drive units, such as 32,767 = maximum frequency. Scaling of the speed feedback from drive units to user (engineering) units and display with suitable units of measure text. Optional reading of an input datalink and scaling of this value from drive raw units to engineering units (such as amperes) for display on the HMI. Reading from the drive and displaying a fault code, and displaying a text fault description based on the fault code. 	<ul style="list-style-type: none"> Using a PowerFlex 753 or PowerFlex 755 drive. The P_PF52x, P_PF753, and P_PF755 Add-On Instructions provide enhanced diagnostics and monitoring for these drive families. Using a PowerFlex 523 or PowerFlex 525 drive. The P_PF52x Add-On Instruction provides enhanced diagnostics and monitoring for these drive families. Using PowerFlex 6000 or PowerFlex 7000 drives. Need to operate the following motors: <ul style="list-style-type: none"> Single-speed motor (running/stopped only) - use the P_Motor instruction. Two-speed motor (fast/slow/stopped only) - use the P_Motor2Spd instruction. Simple reversing motor (forward/reverse/stopped only). - use the P_MotorRev instruction. Motor with multiple discrete speeds - use the P_D4SD instruction. The P_VSD instruction is designed for motors with continuously variable (analog) speed, not multiple discrete speed selections.
E1 Plus Electronic Overload Relay (P_E1PlusE)	<p>This instruction controls and monitors an E1 Plus™ Electronic Overload Relay by using an EtherNet/IP interface module.</p> <p>The instruction monitors the overload relay for warning and trip conditions, displays motor current as a percentage of Full Load amps (% FLA) and percentage motor thermal utilization (% MTU). The instruction also displays a list of the causes of the last five overload trips (trip history).</p> <p>This instruction is provided as a rung import for installation.</p>	<ul style="list-style-type: none"> Use the 193-ETN EtherNet/IP interface to monitor an E1Plus overload relay. Allow for a limited capability for remote reset of overload trips. Provides alarms for trip warning, relay trip, and I/O communication failure. 	<ul style="list-style-type: none"> Do not use this instruction for other Allen-Bradley motor overload relays, such as the E3, E3 Plus, E300™, or 857 series. For the E3 and E3Plus series of motor overload relays, use the P_E3Ovld Add-On Instruction. For the E300 series of motor overload relays, use the P_E300Ovld Add-On Instruction. Other overload relays can be monitored by specific logic or are supported by future Add-On Instructions.

Table 10 - Motors

Process Object	Description	When to Use	When Not to Use
E3/E3Plus Overload Relay (P_E3Ovld)	<p>This instruction controls and monitors the following overload relays:</p> <ul style="list-style-type: none"> 193/592-EC1 193/592-EC2 193/592-EC3 193/592-EC5 <p>The instruction monitors the relay by using a built-in DeviceNet interface or by using a 2100-ENET EtherNet/IP interface. The instruction reports warning and trip conditions, displays motor current as a percentage of Full Load amps (% FLA), and provides commands to initiate a remote trip and a remote trip reset.</p> <p>This instruction is provided as a rung import for installation.</p>	<ul style="list-style-type: none"> Use this instruction for the following capabilities: <ul style="list-style-type: none"> Countdown of time until overload trip can be reset Configurable command to initiate a Remote Test trip and a command to initiate a trip reset Monitors input quality and communication status and provides value and indication of source and quality for the input 	<ul style="list-style-type: none"> Do not use this instruction if you have other Allen-Bradley motor overload relays, such as the E1 Plus, E300, or 857 series: <ul style="list-style-type: none"> For the E1 Plus series of motor overload relays using the 193-ETN EtherNet/IP interface, use the P_E1PlusE Add-On Instruction instead. For the E300 series of motor overload relays, use the P_E300Ovld Add-On Instruction instead. Other overload relays can be monitored by specific logic or are supported by future Add-On Instructions.
E300 Electronic Overload Relay (P_E300Ovld)	<p>This instruction controls and monitors an E300 overload relay via a 193-ECM-ETR EtherNet/IP communication module.</p> <p>This instruction monitors the overload relay for warning and trip conditions. The instruction also displays motor average current and phase currents.</p> <p>The instruction is provided as a rung import for installation.</p>	<ul style="list-style-type: none"> Supports add-on options for the overload relay, including its operator interface, sensors for voltages and ground fault current, and optional digital I/O and analog I/O modules. Provides alarms for trip warning, relay trip, and I/O communication failure. 	<ul style="list-style-type: none"> Do not use this instruction for other Allen-Bradley motor overload relays, such as the E1 Plus, E3, E3 Plus, or 857 series. For the E1 Plus series of motor overload relays that use the 193-ETN EtherNet/IP interface, use the P_E1PlusE Add-On Instruction. For the E3 and E3 Plus series of motor overload relays, use the P_E3Ovld Add-On Instruction. Other overload relays can be monitored by specific logic or supported by future Add-On Instructions.
Runtime and Start Counter (P_RunTime)	<p>This instruction accumulates the total runtime and count of starts for a motor or other equipment. It is a software implementation of the mechanical hour meter that is often mounted in the door of a motor control center (MCC) bucket to show total motor runtime.</p> <p>The runtime and number of starts are variables that are used by maintenance personnel to determine when to perform maintenance activities on the motor or other equipment.</p>	<ul style="list-style-type: none"> Need the functionality of a runtime meter or start counter without having one in your MCC. Need the total runtime, current runtime, maximum runtime, or start count information for a piece of equipment on the operator display, and you do not have monitoring software that provides the information. Equipment monitoring software expects the controller to provide equipment runtime and start values rather than just a running status. 	<ul style="list-style-type: none"> You have advanced software for monitoring equipment runtime that uses the equipment running status as its input.

Table 10 - Motors

Process Object	Description	When to Use	When Not to Use
Restart Inhibit for Large Motor (P_ResInh)	<p>This instruction helps prevent damage to a large motor through repeated starts. The high starting current for a large motor causes considerable heating. The thermal mass of a large motor is much smaller relative to its horsepower and starting current compared to smaller motors. Repeated starts (or start attempts) over a short time overheat the motor windings, potentially damaging the motor permanently.</p> <p>This instruction provides a rule-based state model for restarts and is not intended to model or monitor the motor heating. It cannot replace sensor-based motor monitoring devices. It can, however, be a simple solution to avoid an overstressed motor without the cost (money or controller resources) of more extensive modeling and monitoring.</p>	<ul style="list-style-type: none"> • Have a large motor or other piece of equipment where repeated start/stop cycles or failures to start can damage the equipment. • State model of the P_ResInh instruction is appropriate for limiting the restarts of the equipment (three starts rule, hot/cold model, and so on). • Do not have the sensors or equipment for more advanced motor monitoring or modeling. <p>Note: You can also use the P_Perm instruction, especially if the equipment has additional permissive conditions.</p>	<ul style="list-style-type: none"> • Have more advanced motor monitoring equipment or motor heating models available. Use the advanced equipment to provide a start permissive instead. • Have a small motor that can repeatedly start and stop without damage, or a simple thermal cutout that is provided with the motor is sufficient to help protect it.

Single-speed Motor (P_Motor)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_Motor (Single-speed Motor) Add-On Instruction controls a non-reversing, single-speed motor in various modes and monitors for fault conditions.

Functional Description

The P_Motor instruction provides the following capabilities:

- Operator and Maintenance commands to start and stop the motor and outputs to drive both held and latching starter circuits
- Run feedback and display of actual motor status
- Detection of failure to start and failure to stop
- Permissive conditions to allow starting
- Interlock conditions to stop the motor or prevent starting
- Simulation of a working motor while disabling outputs for use in off-process training or simulation
- Capability for maintenance personnel to take the drive out of service.

IMPORTANT This capability is not a substitute for hard lockout/tagout (LOTO) procedures.

- Input for I/O communication faults
- Alarms for fail to start, fail to stop, interlock trip, and I/O fault
- Operates from Operator, Program, External, Override, Maintenance, and Hand command sources
- An available status for use by automation logic to determine if that logic can control the motor

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_Motor_4.10.00_AOIL5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Operations

This section describes the primary operations for this Add-On Instruction.

IMPORTANT See [Appendix A](#) for Command Sources and Simulation types.

Alarms

This instruction uses the following alarms, which are implemented by using embedded P_Alarm and P_Gate Add-On Instructions.

Alarm Name	P_Alarm Name	P_Gate Name	Description
Fail to Start	FailToStart	None	Raised when the motor has and is using run feedback, an attempt is made to start the motor, and the run feedback does not indicate that the motor is running within the configured time. If Fail to Start is configured as a shed fault, the motor is stopped and a reset is required to start the motor.
Fail to Stop	FailToStop	None	Raised when the motor has and is using run feedback, an attempt is made to stop the motor, and the run feedback does not indicate that the motor stopped within the configured time.
Interlock Trip	IntlkTrip	None	Raised when the motor is running and an interlock 'not OK' condition causes the motor to stop. If interlocks are not bypassed, a bypassable interlock or a non-bypassable interlock 'not OK' condition initiates an interlock trip. If interlocks are bypassed, only a non-bypassable interlock 'not OK' condition initiates an interlock trip.
I/O Fault	IOFault	None	Raised when the Inp_IOFault input is true. This input is used to indicate to the instruction that a communication failure has occurred for its I/O. If the I/O Fault is configured as a shed fault, the motor is stopped and not permitted to start until reset.


Parameters of the P_Alarm object can be accessed by using the following convention: [P_Alarm Name].[P_Alarm Parameter].

The Fail to Start and Fail to Stop status and alarms have a configurable delay to allow the run feedback time to align with the commanded output. This delay provides time for the motor to start or stop.

The Fail to Start and I/O fault conditions can be configured to alarm only, or to de-energize the motor (shed). If one of these conditions stops the motor, a reset is required to run.

Simulation

When P_Motor is in simulation, the instruction keeps its outputs de-energized and simulates a working motor.

Set the Inp_Sim parameter in the Controller to '1' to enable simulation. When the motor is in simulation, the Simulation or Loopback Test icon  is displayed at the top left of the Operator faceplate.

When in simulation, the object responds to start, stop and jog commands as if a working motor were being controlled. The configuration parameter Cfg_SimFdbkT sets the number of seconds for the simulation to echo back the running or stopped status. For example, if Cfg_SimFdbkT is set to 2.0 seconds and you stop the motor, the simulation will show a "Stopping" status for 2 seconds before showing "Stopped."

When you have finished in simulation, set the Inp_Sim parameter in the Controller to '0' to return the motor to normal operation.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	Processing for EnableIn False (false rung) is handled the same as if the motor were taken out of service by Command. The motor outputs are de-energized and the motor is shown as Program Out of Service on the HMI. All alarms are cleared.
Powerup (prescan, first scan)	Any commands that are received before first scan are discarded. The motor is de-energized and treated as if it were commanded to stop. Embedded P_CmdSrc and P_Alarm instructions are handled in accordance with their standard powerup procedures. See the Reference Manuals for the P_CmdSrc and P_Alarm instructions for details.
Postscan (SFC transition)	No SFC postscan logic is provided.

For more information, see the Logix 5000™ Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#).

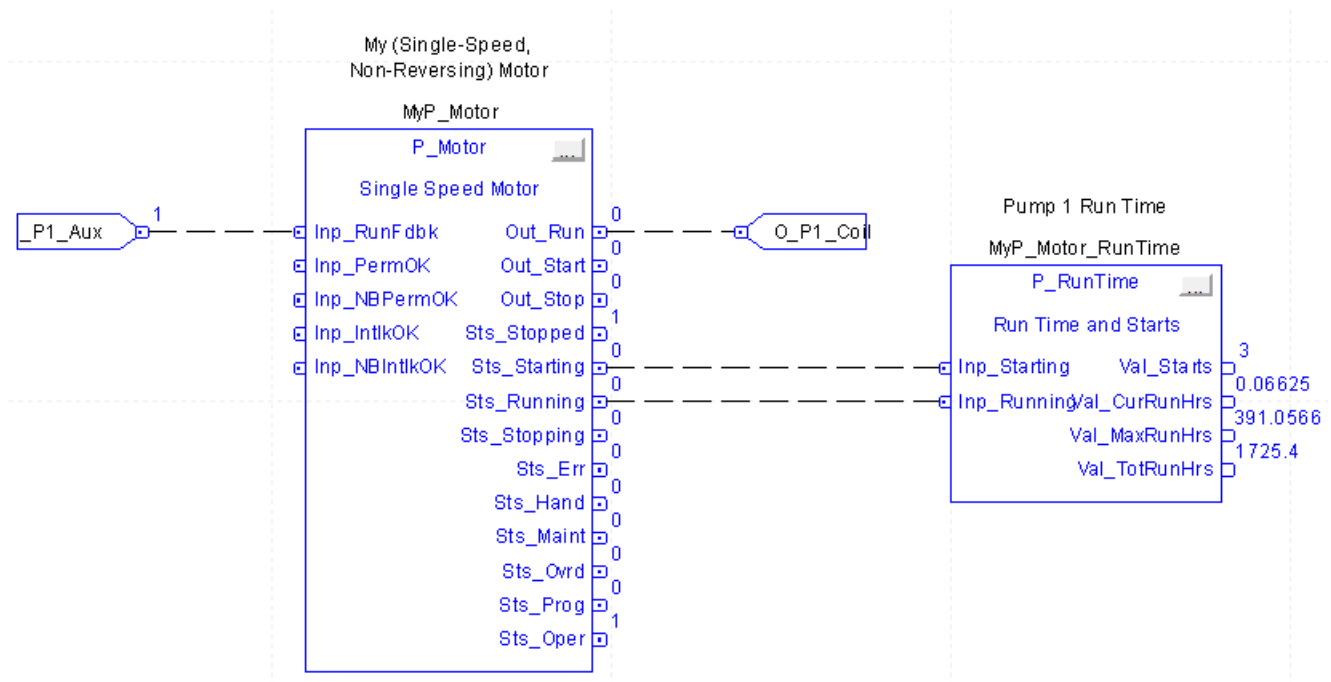
Programming Example

The following is a simple example of P_Motor.

Boolean parameter I_P1_Aux is used as an input. A single output, O_P1_Coil, is wired to energize the starter coil (1) to start or (0) stop the motor.

To use input I_P1_Aux, both Cfg_HasRunFdbk and Cfg_UseRunFdbk must be set to 1 (the default for both of these parameters is 0).

Here, the P_Motor block is connected to a P_RunTime block to demonstrate the simplicity of the interaction of these Add-On Instructions.



Two-speed Motor (P_Motor2Spd)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_Motor2Spd (Two-speed Motor) Add-On Instruction controls a non-reversing, two-speed motor (fast/slow/stopped) in various modes and monitors for fault conditions.

Functional Description

The P_Motor2Spd instruction provides the following capabilities:

- Controls outputs to start a two-speed motor fast or slow and stop the motor.
- Motors run feedback (optional) and display actual motor status
- Detects failure to start or stop and generates appropriate alarms
- Monitors permissive conditions to allow starting. Separate permissives are provided to allow running fast and running slow
- Monitors interlock conditions to stop the motor or prevent starting, and alarms when an interlock trips the motor
- Provides alarms for Failure to Start, Failure to Stop, Interlock Trip, and I/O fault
- Capability for maintenance personnel to take the drive out of service.

IMPORTANT This capability is not a substitute for hard lockout/tagout (LOTO) procedures.

- Provides for simulation of a working motor while disabling outputs, for use in off-process training, testing, or simulation
- Monitors I/O communication, and alarms and shuts down on a communication fault
- Operates from Operator, Program, External, Override, Maintenance, and Hand command sources.
- Provides an available status for use by automation logic to determine if other program logic can start and stop the motor

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is

defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

- The P_Motor2Spd_4.10.**00**_AOIL5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Operations

This section describes the primary operations for this Add-On Instruction.

IMPORTANT See [Appendix A](#) for Command Sources and Simulation types.

Alarms

This instruction uses the following alarms, which are implemented by using embedded P_Alarm and P_Gate Add-On Instructions.

Alarm Name	P_Alarm Name	P-Gate Name	Description
Fail to Start	FailToStart	None	Raised when the motor has and is using run feedback, an attempt is made to start the motor, and the run feedback does not indicate that the motor is running at the correct speed within the configured time. If Fail to Start is configured as a shed fault, the motor is stopped and a reset is required to start the motor.
Fail to Stop	FailToStop	None	Raised when the motor has and is using run feedback, an attempt is made to stop the motor, and the run feedback does not indicate that the motor stopped within the configured time.
Interlock Trip	IntlkTrip	None	Raised when the motor is running and an interlock 'not OK' condition causes the motor to stop. If interlocks are not bypassed, a bypassable interlock or a non-bypassable interlock 'not OK' condition initiates an interlock trip. If interlocks are bypassed, only a non-bypassable interlock 'not OK' condition initiates an interlock trip.
I/O Fault	IOFault	None	Raised when the Inp_IOFault input is true. This input is used to indicate to the instruction that a communication failure has occurred for its I/O. If the I/O Fault is configured as a shed fault, the motor is stopped and not permitted to start until reset.


Parameters of the P_Alarm object can be accessed by using the following convention: [P_Alarm Name].[P_Alarm Parameter].

The Fail to Start and Fail to Stop status and alarms have a configurable delay to allow the run feedback time to align with the commanded output. This delay provides time for the motor to start or stop.

The Fail to Start and I/O fault conditions can be configured to alarm only, or to de-energize the motor (shed). If one of these conditions stops the motor, a reset is required to run.

Simulation

When P_Motor2Spd is in simulation, the instruction keeps its outputs de-energized and simulates a working motor.

Set the Inp_Sim parameter in the Controller to '1' to enable simulation. When the motor is in simulation, the Simulation or Loopback Test icon  is displayed at the top left of the Operator faceplate.

When in simulation, the object responds to start, stop and jog commands as if a working motor were being controlled. The configuration parameter Cfg_SimFdbkT sets the number of seconds for the simulation to echo back the running or stopped status. For example, if Cfg_SimFdbkT is set to 2.0 seconds and you stop the motor, the simulation will show a "Stopping" status for 2 seconds before showing "Stopped."

When you have finished in simulation, set the Inp_Sim parameter in the Controller to '0' to return the motor to normal operation.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	Processing for EnableIn False (false rung) is handled the same as if the motor were taken out of service by Command. The motor outputs are de-energized and the motor is shown as Program Out of Service on the HMI. All alarms are cleared.
Powerup (prescan, first scan)	Any commands that are received before first scan are discarded. The motor is de-energized and treated as if it were commanded to stop. Embedded P_CmdSrc and P_Alarm instructions are handled in accordance with their standard powerup procedures. See the Reference Manuals for the P_CmdSrc and P_Alarm instructions for details.
Postscan (SFC transition)	No SFC postscan logic is provided.

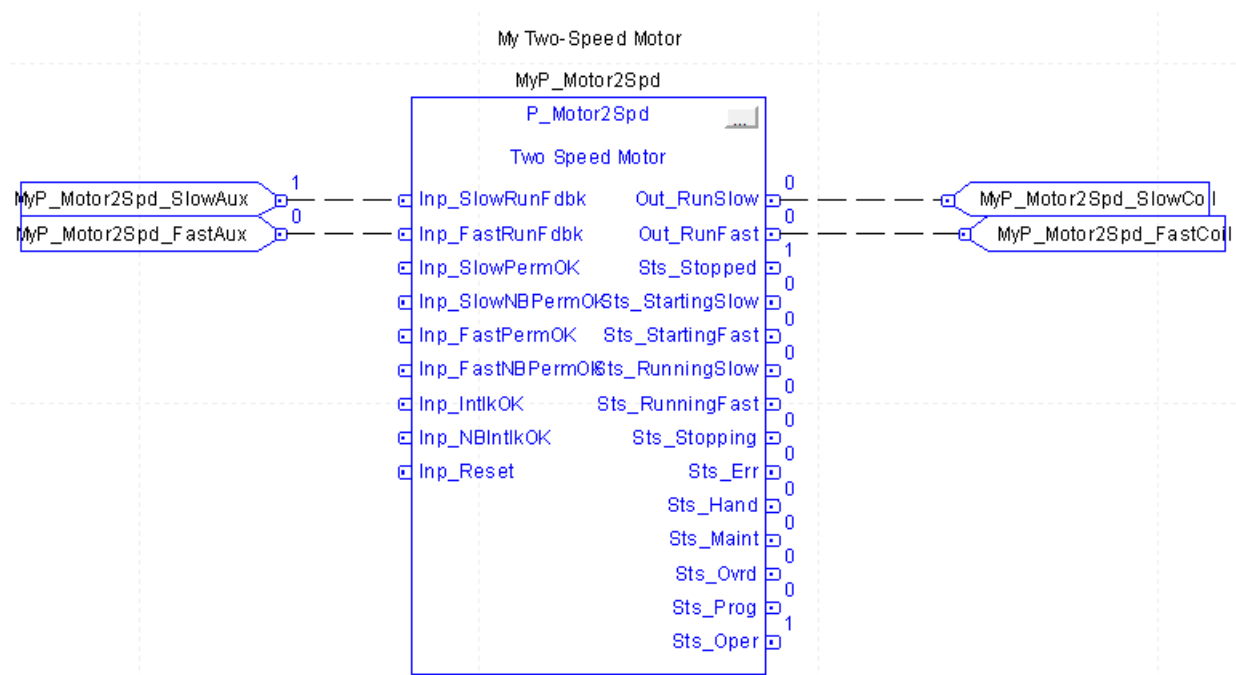
For more information, see the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#).

Programming Example

The following is a simple example of P_Motor2Spd.

Boolean parameters MyP_Motor2Spd_FastAux and MyP_Motor2Spd_SlowAux are used as inputs. Outputs MyP_Motor2Spd_SlowCoil and MyP_Motor2Spd_FastCoil are wired to energize the appropriate starter circuits.

To use inputs MyP_Motor2Spd_FastAux and MyP_Motor2Spd_SlowAux, both Cfg_HasRunFdbk and Cfg_UseRunFdbk must be set to 1 (the default for both of these parameters is 0).



Reversing Motor (P_MotorRev)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_MotorRev (Reversing Motor) Add-On Instruction controls a reversing motor (forward/reverse/stopped) in various modes and monitors for fault conditions.

Functional Description

The P_MotorRev instruction provides the following capabilities:

- Controls outputs to start a reversing motor in the forward or reverse direction or stop the motor.
- run feedback (optional) and display actual motor status.
- Detects failure to start or stop and generates appropriate alarms.
- Monitors permissive conditions to allow starting. Separate permissives are provided to allow running forward and running reverse.
- Monitors interlock conditions to stop the motor or prevent starting, and alarms when an interlock trips the motor.
- Provides alarms for Failure to Start, Failure to Stop, Interlock Trip, and I/O fault.
- Capability for maintenance personnel to take the drive out of service.

IMPORTANT This capability is not a substitute for hard lockout/tagout (LOTO) procedures.

- Provides for simulation of a working motor while disabling outputs, for use in off-process training, testing, or simulation.
- Monitors I/O communication, and alarms and shuts down on a communication fault.
- Operates from Operator, Program, External, Override, Maintenance, and Hand command sources.
- Provides an available status for use by automation logic to determine if other program logic can start and stop the motor.

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is

defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

- The P_MotorRev_4.10.00_AOIL5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Operations

This section describes the primary operations for Add-On Instructions.

IMPORTANT See [Appendix A](#) for Command Sources and Simulation types.

Alarms

This instruction uses the following alarms, which are implemented by using embedded P_Alarm and P_Gate Add-On Instructions.

Alarm Name	P_Alarm Name	P-Gate Name	Description
Fail to Start	FailToStart	None	Raised when the motor has and is using run feedback, an attempt is made to start the motor, and the run feedback does not indicate that the motor is running in the correct direction within the configured time. If Fail to Start is configured as a shed fault, the motor is stopped and a reset is required to start the motor.
Fail to Stop	FailToStop	None	Raised when the motor has and is using run feedback, an attempt is made to stop the motor, and the run feedback does not indicate that the motor stopped within the configured time.
Interlock Trip	IntlkTrip	None	Raised when the motor is running and an interlock 'not OK' condition causes the motor to stop. If interlocks are not bypassed, a bypassable interlock or a non-bypassable interlock 'not OK' condition initiates an interlock trip. If interlocks are bypassed, only a non-bypassable interlock 'not OK' condition initiates an interlock trip.
I/O Fault	IOFault	None	Raised when the Inp_IOFault input is true. This input is used to indicate to the instruction that a communication failure has occurred for its I/O. If the I/O Fault is configured as a shed fault, the motor is stopped and not permitted to start until reset.


Parameters of the P_Alarm object can be accessed by using the following convention: [P_Alarm Name].[P_Alarm Parameter].

The Fail to Start and Fail to Stop status and alarms have a configurable delay to allow the run feedback time to align with the commanded output. This delay provides time for the motor to start or stop.

The Fail to Start and I/O fault conditions can be configured to alarm only, or to de-energize the motor (shed). If one of these conditions stops the motor, a reset is required to run.

Simulation

When P_MotorRev is in simulation, the instruction keeps its outputs de-energized and simulates a working motor.

Set the Inp_Sim parameter in the Controller to '1' to enable simulation. When the motor is in simulation, the Simulation or Loopback Test icon  is displayed at the top left of the Operator faceplate.

When in simulation, the object responds to start, stop and jog commands as if a working motor were being controlled. The configuration parameter Cfg_SimFdbkT sets the number of seconds for the simulation to echo back the running or stopped status. For example, if Cfg_SimFdbkT is set to 2.0 seconds and you stop the motor, the simulation will show a "Stopping" status for 2 seconds before showing "Stopped."

When you have finished in simulation, set the Inp_Sim parameter in the Controller to '0' to return the motor to normal operation.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	Processing for EnableIn False (false rung) is handled the same as if the motor were taken out of service by Command. The motor outputs are de-energized and the motor is shown as Program Out of Service on the HMI. All alarms are cleared.
Powerup (prescan, first scan)	Any commands that are received before first scan are discarded. The motor is de-energized and treated as if it were commanded to stop. Embedded P_CmdSrc and P_Alarm instructions are handled in accordance with their standard powerup procedures. See the reference manuals for the P_CmdSrc and P_Alarm instructions for details.
Postscan (SFC transition)	No SFC postscan logic is provided.

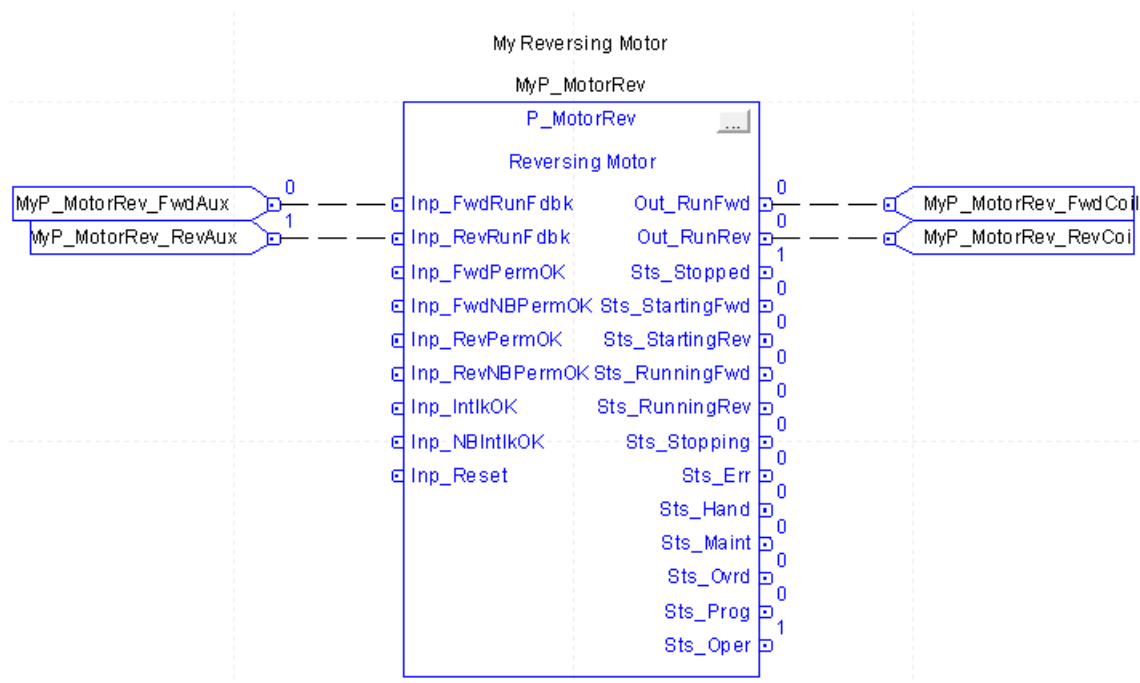
For more information, see the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#).

Programming Example

The following is a simple example of P_MotorRev.

Boolean parameters MyP_MotorRev_FwdAux and MyP_MotorRev_RevAux are used as feedback inputs. Outputs MyP_MotorRev_FwdCoil and MyP_MotorRev_RevCoil are outputs to the motor starter.

To use inputs MyP_MotorRev_FwdAux and MyP_MotorRev_RevAux, both Cfg_HasRunFdbk and Cfg_UseRunFdbk must be set to 1 (the default for both of these parameters is 0).



Hand-operated Motor (P_MotorHO)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_MotorHO (Hand-operated Motor) Add-On Instruction monitors a locally-controlled (hand-operated) motor.

Functional Description

The P_MotorHO instruction provides the following capabilities:

- Monitors the run feedback signals from a single-speed, two-speed, or reversing motor and display actual motor status.
- The ability to trip (de-energize) the motor (optional). The program (via program commands) or the operator (via the HMI faceplate) can trip the motor any time it is running. Interlocks can also be used to trip the motor.

The optional trip function provides the following capabilities:

- Detects failure to stop when tripped and generate an appropriate alarm.
- Monitors interlock conditions to trip the motor, and alarm when an interlock stops a running motor.
- Provides for simulation of a working motor while disabling the trip output, for use in off-process training, testing, or simulation.
- Monitors I/O communication, and alarm (and trip if the Shed On I/O Fault function is enabled) on a communication fault.

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_MotorHO_4.10.00_AOI.L5X Add-On Instruction must be imported into the controller project be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Operations

This section describes the primary operations for this Add-On Instruction.

Alarms

This instruction uses the following alarms, which are implemented by using embedded P_Alarm and P_Gate Add-On Instructions.


Alarm Name	P_Alarm Name	P-Gate Name	Description
Interlock Trip	IntlkTrip	None	Raised when the motor is running, the optional trip function is used, and an interlock 'not OK' condition triggers the trip function to stop the motor. If interlocks are not bypassed, a bypassable interlock or a non-bypassable interlock 'not OK' condition initiates an interlock trip. If interlocks are bypassed, only a non-bypassable interlock 'not OK' condition initiates an interlock trip.
I/O Fault	IOFault	None	Raised when the Inp_IOFault input is true. This input is used to indicate to the instruction that a communication failure has occurred for its I/O. If the I/O Fault is configured as a shed fault and the optional trip function is used, the trip output is triggered until reset.
Trip Failure	TripFail	None	Raised is the motor has and is using the optional trip feature, an attempt is made to trip (stop) the motor, and the run feedbacks show that the motor did not stop within the configured fail to trip time.

Parameters of the P_Alarm object can be accessed by using the following convention: [P_Alarm Name].[P_Alarm Parameter].

The trip fail alarm has a configurable delay to allow the run feedback time to show that the motor actually stopped (when tripped) before raising an alarm.

Simulation

When P_MotorHO is in simulation, the instruction keeps the its output de-energized and simulates monitoring a working motor.

Set the Inp_Sim parameter in the Controller to '1' to enable simulation. When the motor is in simulation, the Simulation or Loopback Test icon  is displayed at the top left of the Operator faceplate.

When in simulation, if the parameter Inp_SimRunFastFwd is set, the simulated motor is set to the running state (for a single-speed non-reversing motor), or to the running fast state (for a two-speed motor) or to the running forward state (for a reversing motor). The Trip function is simulated, and the configuration parameter Cfg_SimFdbkT sets the number of seconds for the simulation to echo back the tripped status. For example, if Cfg_SimFdbkT is set to 2.0 seconds and

you trip the motor, the simulation will show a “Tripping” status for 2 seconds before showing “Stopped”.

When you have finished in simulation, set the Inp_Sim parameter in the Controller to ‘0’ to return the motor to normal operation.

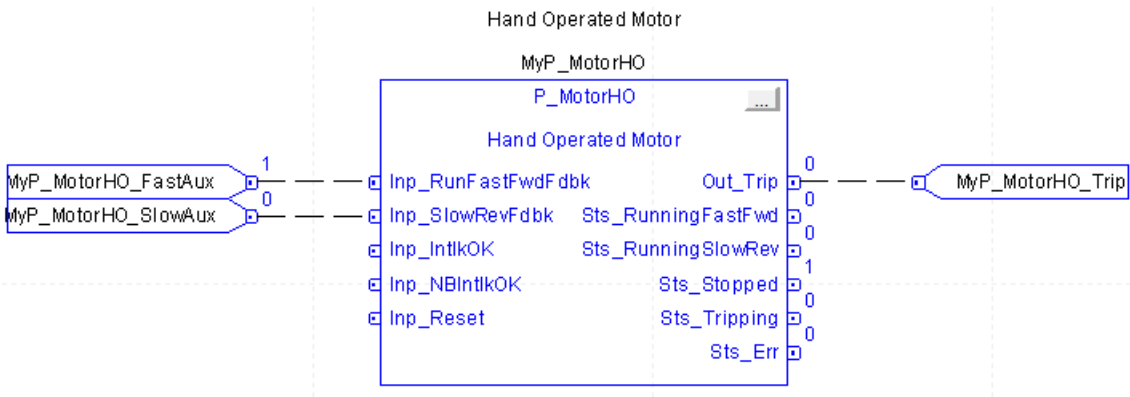
Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn false (false rung)	Handled the same as if the trip function option were disabled. The trip output (Out_Trip) is de-energized. The instruction reverts to monitoring only the current state of the motor. All alarms are cleared.
Powerup (pre-scan, first scan)	Any commands that are received before first scan are discarded. The Trip output is de-energized to prevent a nuisance trip on first scan. Embedded P_Alarm instructions are handled in accordance with their standard powerup procedures. See the reference manual for the P_Alarm instructions for more information.
Postscan (SFC transition)	No SFC postscan logic is provided.

Programming Example

The following is a simple example of P_MotorHO. Boolean parameters MyP_MotorHO_FastAux and MyP_MotorHO_SlowAux are used as inputs. A single output, MyP_MotorHO_Trip, is wired to an output that trips the motor circuit.



PowerFlex 523/525 Drives (P_PF52x)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_PF52x (PowerFlex 523/525 Variable-Frequency Drives) object is used to control and monitor a PowerFlex 523 Variable-Frequency Drive with optional EtherNet/IP Interface. The P_PF52x can also control and monitor a PowerFlex 525 Variable-Frequency Drive with embedded or optional add-on EtherNet/IP Interface.

Functional Description

The P_PF52x instruction provides the following capabilities:

- Starting, stopping, jogging of the drive, and setting speed reference and direction
- Monitoring of run feedback, display of actual drive status, including acceleration, deceleration, direction, and speed
- Detection of Failure to Start, Failure to Stop, and Drive Fault
- Monitoring of Permissive conditions to allow starting
- Monitoring of Interlock conditions to stop/prevent starting
- Simulation, providing feedback of a working drive while the outputs are disabled
- Capability for maintenance personnel to take the drive out of service.

IMPORTANT This capability is not a substitute for hard lockout/tagout (LOTO) procedures.

- Monitoring I/O communication faults
- Alarms for Fail to Start, Fail to Stop, Interlock Trip, Drive Fault, and I/O Fault
- Option to reset faults and alarms automatically when an operator commands the motor to start or stop
- Supports HMI 'breadcrumbs' for Alarm Inhibited, Bad Configuration, Not Ready, and Maintenance Bypass Active
- 'Available' status for use by automation logic to know whether a motor can be controlled by other objects

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This code lets you create your

own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_PF52x_4.10.00_RUNG.L5X rung import must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Required Drive Configuration

Be certain to configure the drive Datalinks as follows:

IMPORTANT	'User Choice' Datalinks are not used by this Add-On Instruction and can be left unused or configured for your application.
------------------	--

- Input Assembly
 - Drive Status (standard)
 - Output Frequency (standard)
 - Datalinks:
 1. Fault 1 Code (Par 007)
 2. Output Current (Par 003)
 3. Output Power (Par 017)
 4. User choice
- Output Assembly
 - Drive Logic Command (standard)
 - Frequency Command (Speed Reference) (standard)
 - Datalinks:

All four output datalinks are user choice

InOut Structure

InOut parameters are used to link the Add-On Instruction to external tags that contain necessary data for the instruction to operate. These external tags must be of the data type shown.

Tag Name	Data Type	Description
Inp	P_PF52x_Inp	Common part of PowerFlex 52x input assembly.
Out	P_PF52x_Out	Common part of PowerFlex 52x output assembly.
Ref_FaultCodeList	P_DescList[1]	Array tag that contains a list of fault codes (DINT) and their descriptions (STRING).

When the P_PF52x instruction is instantiated using the RUNG import, the “PF525_FaultCodeList” tag shown in the following image, is included in the import and created if it does not exist. This tag is also included in the Template applications that are included in the Library download.

Scope: ProcessObjects_4		Show: All Tags		Enter Name Filter...	
Name	Value	Force Mask	Style	Data Type	Description
+ PF40E:O	{...}	{...}		AB.PowerFlex...	
+ PF75x_FaultCodeList	{...}	{...}		P_DescList[21]	PowerFlex 753 / 755 Fault Codes and Descriptions
+ PF525_FaultCodeList	{...}	{...}		P_DescList[61]	PowerFlex 525 VFD Fault Codes and Descriptions
+ PF700S_FaultCodeList	{...}	{...}		P_DescList[80]	PowerFlex 700S VFD Fault Codes and Descriptions
+ PF753_EtherNetIP:I	{...}	{...}		AB.PowerFlex...	

To display fault code messages in P_PF52x, enter the name of the Fault Code List tag (first column) in the P_PF52x Ref_FaultCodeList parameter.

Each fault code list has preset codes and descriptions for translating fault code numbers that are received from the drive to human-readable drive fault descriptions.

Scope: ProcessObjects_4		Show: All Tags		Enter Name Filter...	
Name	Value	Force Mask	Style	Data Type	Description
- PF525_FaultCodeList	{...}	{...}		P_DescList[61]	PowerFlex 525 VFD Fault Codes and Descriptions
- PF525_FaultCodeList[0]	{...}	{...}		P_DescList	PowerFlex 525 VFD Fault Codes and Descriptions Code / Description List Entry
+ PF525_FaultCodeList[0].Code	0		Decimal	DINT	Code / Description List Entry Code for which to look up Description
+ PF525_FaultCodeList[0].Desc	'Check drive manual ...	{...}		STRING_40	Code / Description List Entry Description for given Code
- PF525_FaultCodeList[1]	{...}	{...}		P_DescList	PowerFlex 525 VFD Fault Codes and Descriptions Code / Description List Entry
+ PF525_FaultCodeList[1].Code	2		Decimal	DINT	Code / Description List Entry Code for which to look up Description
+ PF525_FaultCodeList[1].Desc	'Auxiliary Input'	{...}		STRING_40	Code / Description List Entry Description for given Code
- PF525_FaultCodeList[2]	{...}	{...}		P_DescList	PowerFlex 525 VFD Fault Codes and Descriptions Code / Description List Entry
+ PF525_FaultCodeList[2].Code	3		Decimal	DINT	Code / Description List Entry Code for which to look up Description
+ PF525_FaultCodeList[2].Desc	'Power Loss'	{...}		STRING_40	Code / Description List Entry Description for given Code
- PF525_FaultCodeList[3]	{...}	{...}		P_DescList	PowerFlex 525 VFD Fault Codes and Descriptions Code / Description List Entry
+ PF525_FaultCodeList[3].Code	4		Decimal	DINT	Code / Description List Entry Code for which to look up Description
+ PF525_FaultCodeList[3].Desc	'Under Voltage'	{...}		STRING_40	Code / Description List Entry Description for given Code

For a list of fault codes, see the PowerFlex 520-series Adjustable Frequency AC Drives User Manual, publication [520-UM001](#).

Operations

This section describes the primary operations for this Add-On Instruction.

IMPORTANT See [Appendix A](#) for Command Sources and Simulation types.

Alarms


This instruction uses the following alarms, which are implemented by using embedded P_Alarm and P_Gate Add-On Instructions.

Alarm Name	P_Alarm Name	P-Gate Name	Description
Drive Fault	DriveFault	None	Raised when the drive detects a fault and sets its Faulted status bit. Check the Fault Code and description to determine the cause. Issuing a Reset of this object causes a Clear Fault command to be sent to the drive in an attempt to clear the fault.
Fail to Start	FailToStart	None	Raised when the drive has and is using run feedback, an attempt is made to start the drive, and the run feedback does not indicate that the drive is running within the configured time. If Fail to Start is configured as a shed fault, the drive is stopped and a reset is required to start the drive.
Fail to Stop	FailToStop	None	Raised when the drive has and is using run feedback, an attempt is made to stop the drive, and the run feedback does not indicate that the drive stopped within the configured time.
Interlock Trip	IntlkTrip	None	Raised when the drive is running and an interlock 'not OK' condition causes the drive to stop. If interlocks are not bypassed, a bypassable interlock or a non-bypassable interlock 'not OK' condition initiates an interlock trip. If interlocks are bypassed, only a non-bypassable interlock 'not OK' condition initiates an interlock trip.
I/O Fault	IOFault	None	Raised when the Inp_IOFault input is true. This input is used to indicate to the instruction that a communication failure has occurred for its I/O. If the I/O Fault is configured as a shed fault, the drive is stopped and not permitted to start until reset.

Parameters of the P_Alarm object can be accessed by using the following convention: [P_Alarm Name].[P_Alarm Parameter].

Simulation

When P_PF52x is in simulation, the instruction keeps its outputs de-energized and simulates a working drive.

Set the Inp_Sim parameter in the Controller to '1' to enable simulation. When the drive is in simulation, the Simulation or Loopback Test icon  is displayed at the top left of the Operator faceplate.

When in simulation, the object responds to start, stop and jog commands and speed reference changes as if a working drive were being controlled. The configuration parameter Cfg_SimRampT sets the amount of time the simulation takes to ramp from zero speed to maximum and vice versa. The stopped status is asserted when the simulated speed feedback is zero, and running/jogging status is asserted when the simulated speed is greater than zero. There are also configuration bits to determine how speed feedback is calculated based on the speed reference. See the faceplate to see how these configuration bits perform the scaling.

When you have finished in simulation, set the Inp_Sim parameter in the Controller to '0' to return the motor to normal operation.

Execution

The following table explains the handling of instruction execution conditions.

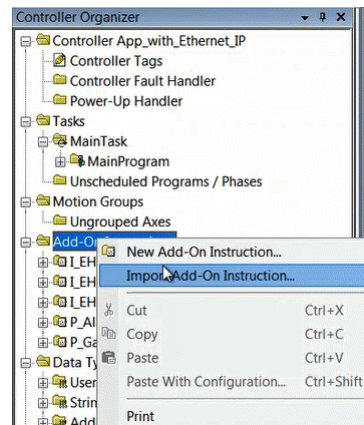
Condition	Description
EnableIn False (false rung)	Processing for EnableIn False (false rung) is handled the same as if the drive were taken out of service by Command. The drive outputs are de-energized and the drive is shown as Program Out of Service on the HMI. All alarms are cleared.
Powerup (prescan, first scan)	Processing of modes and alarms on prescan and Powerup is handled by the embedded P_CmdSrc and P_Alarm Add-On Instructions. See their specifications for details. On Powerup, the drive is treated as if it had been Commanded to Stop.
Postscan (SFC transition)	No SFC Postscan logic is provided.

Programming Example

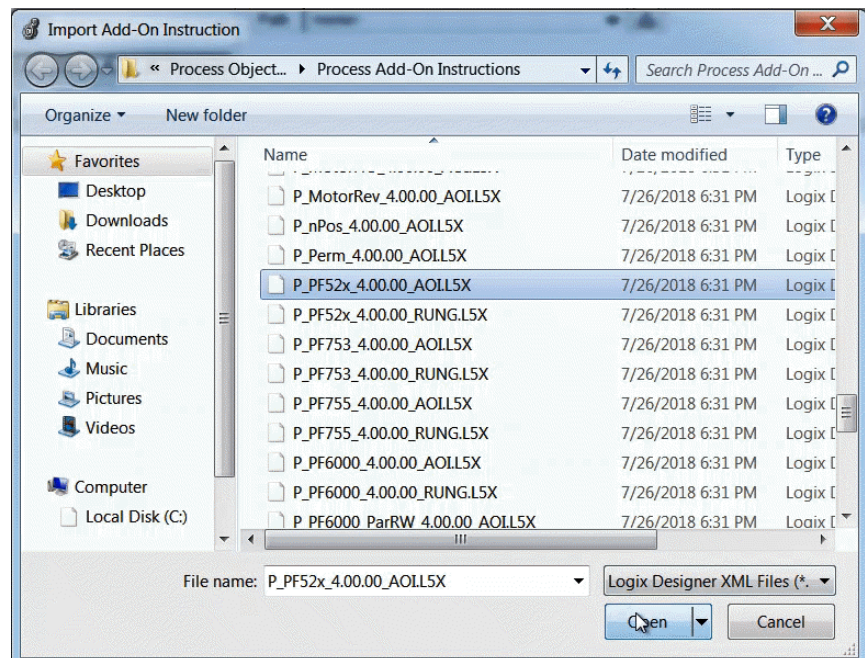
Import Device Add-On Instruction

This procedure imports the definitions for the device Add-On Instruction. It is only necessary to import the definitions once per controller.

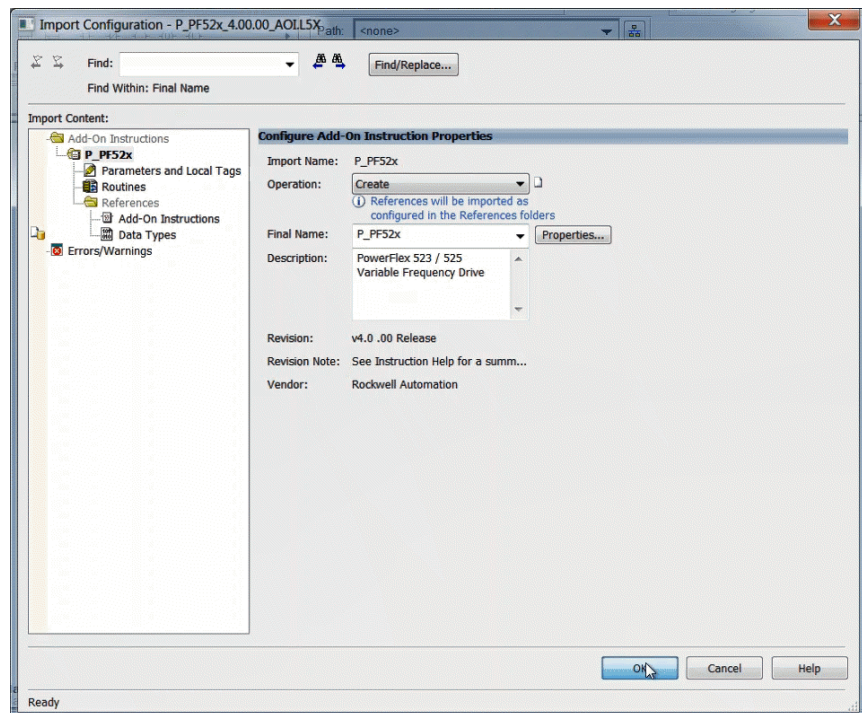
1. In the target Controller Organizer, right-click on Add-On Instructions and choose Import Add-On Instruction.



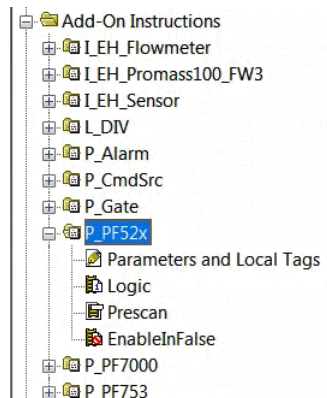
2. Navigate to the folder that contains the device Add-On Instructions, select the appropriate instruction, and click open.



3. Click OK in the Import Configuration window.

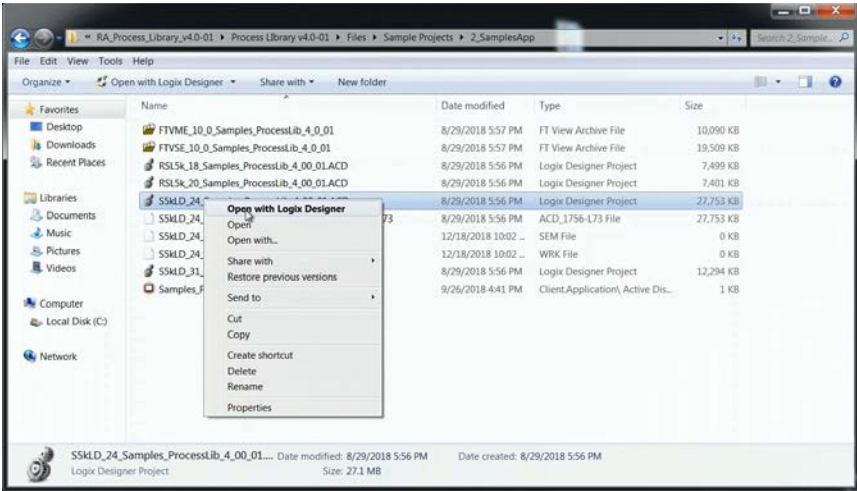


4. The Add-On Instruction is then added to the Controller Organizer.

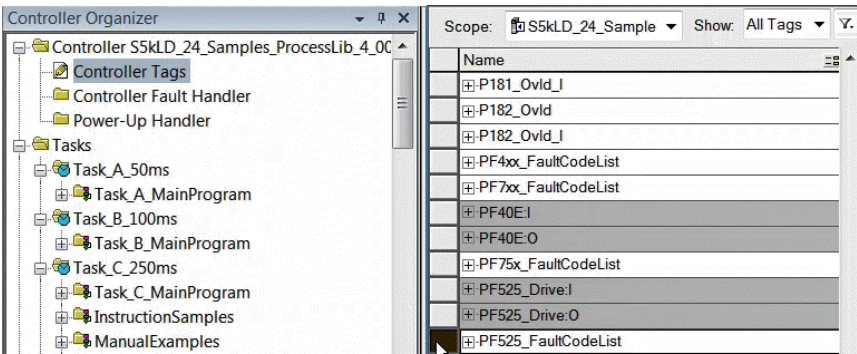


Copy Fault Code Tags

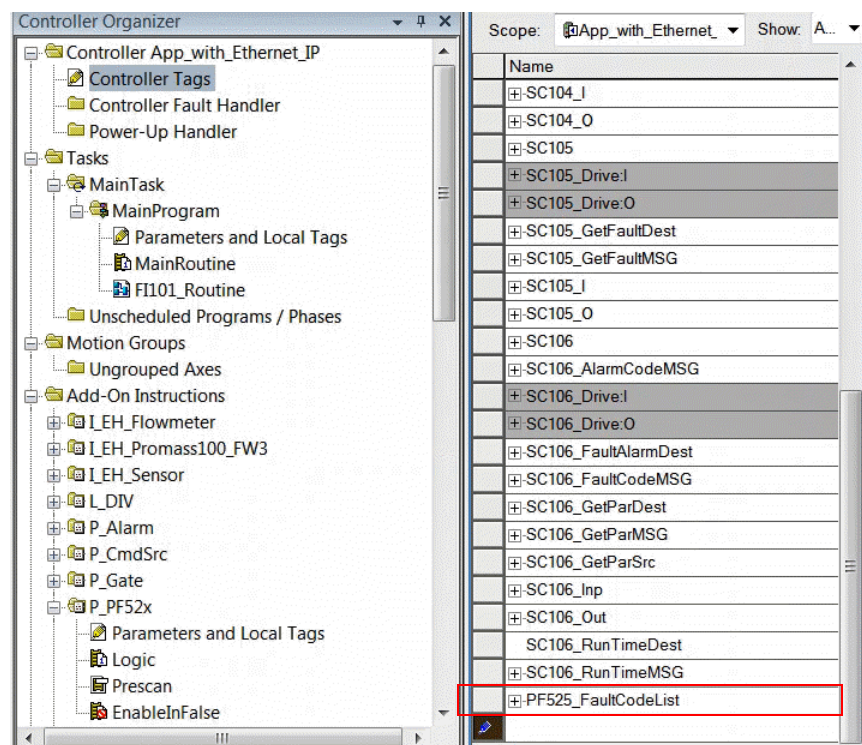
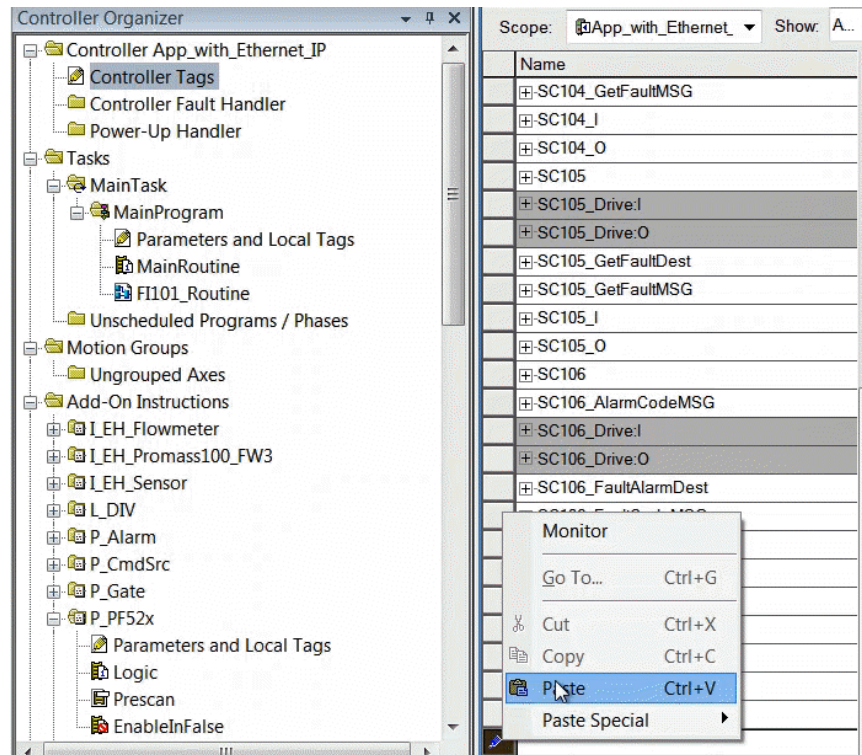
1. Open a Project in the Files>Sample Projects>2_SamplesApp Project folder.



2. In the Sample Application, Click Controller Tags in the Controller Organizer. Right Click on the PF525_FaultCodeList and choose Copy.

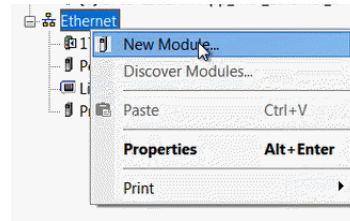


3. In the main project, Click Controller Tags in the Controller Organizer. Right Click at the bottom of the tag list and choose Paste.

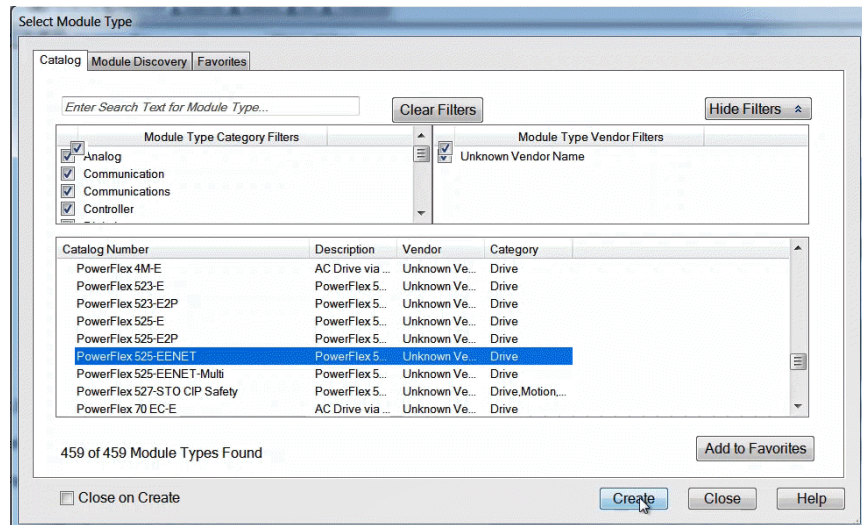


Add the Device to the I/O Tree and Configure the Device

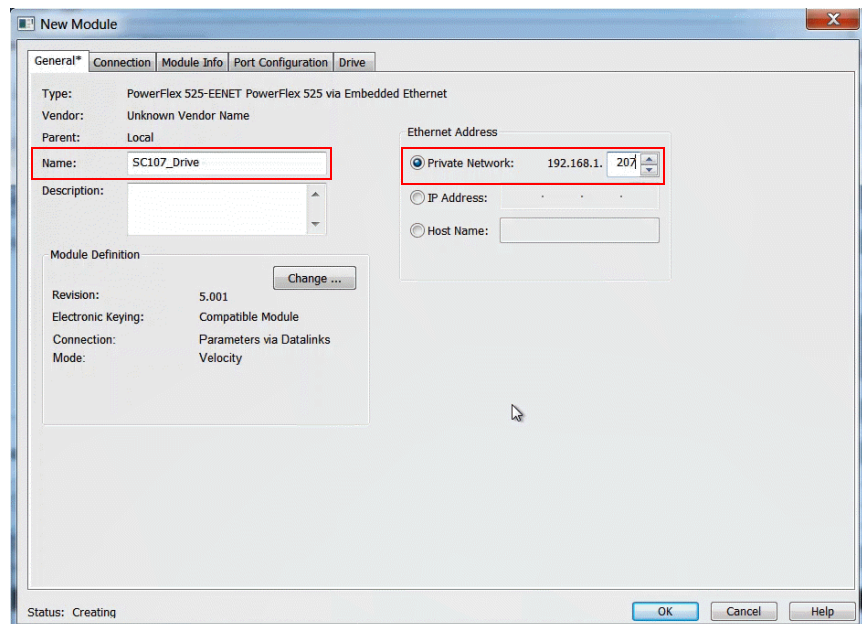
1. In your target application, right-click the Ethernet network in the Controller Organizer and choose New Module.



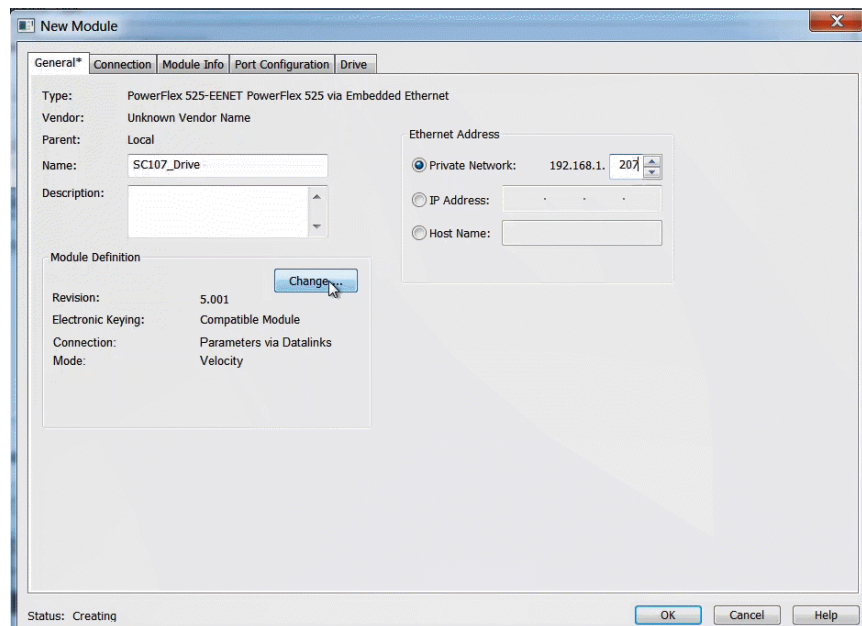
2. Select the Module Type and click Create.



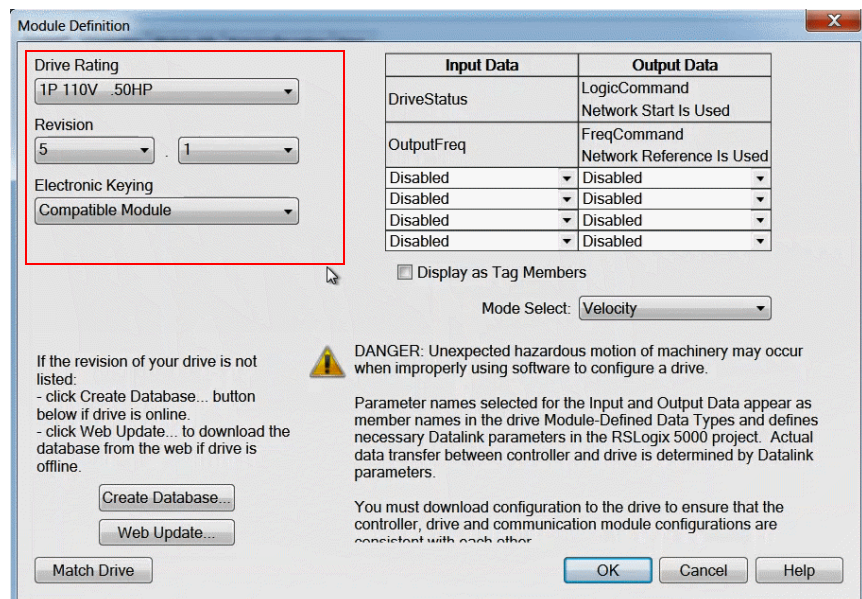
3. Change the device name and IP address to match the specifications of your project.



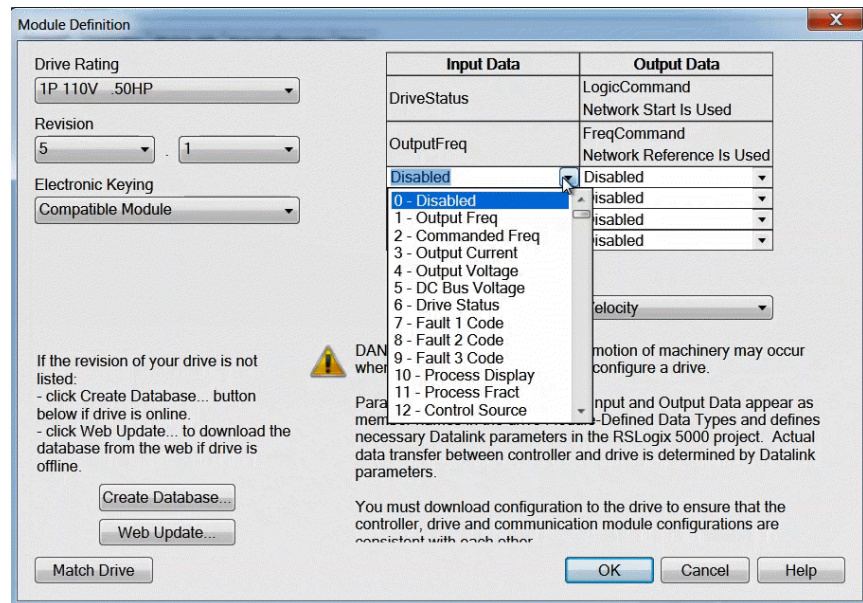
4. Click Change in the Module Definition section.



5. Change the information for Drive Rating, Revision, and Electronic Keying to match the specifications of your project.



6. In the Input Data column, click the pull-down menu, choose the parameter for each Datalink and click OK. You will return to the Module Definition dialog box after each Datalink.



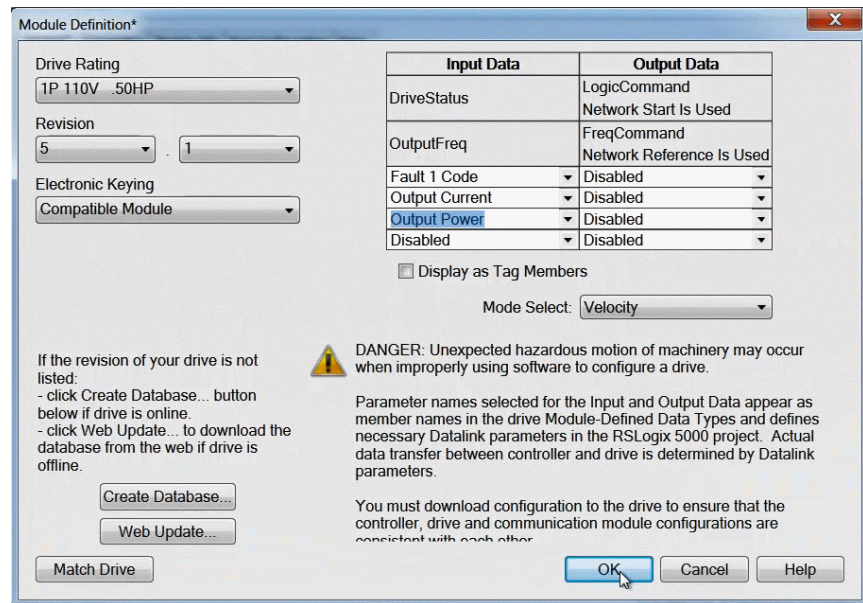
Repeat step 6 for each Datalink.

The required DataLinks to add to your project are:

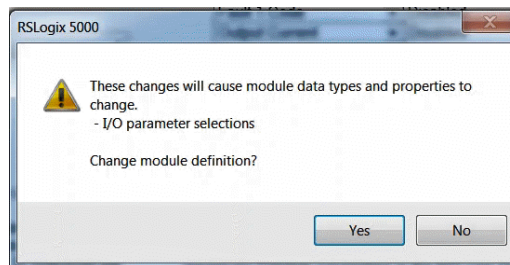
- Input Assembly
 - Drive Status (standard)
 - Output Frequency (standard)
 - Datalinks:
 1. Fault 1 Code (Par 007)
 2. Output Current (Par 003)
 3. Output Power (Par 017)
 4. User choice
- Output Assembly
 - Drive Logic Command (standard)
 - Frequency Command (Speed Reference) (standard)
 - Datalinks:

All four output datalinks are user choice

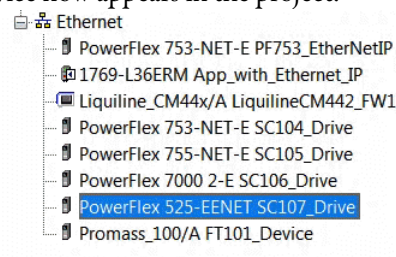
7. Once all Datalinks are provisioned click OK in the Module Definition dialog box.



8. Click YES in the confirmation dialog box to accept the changes to the datalinks.



9. The selected device now appears in the project.

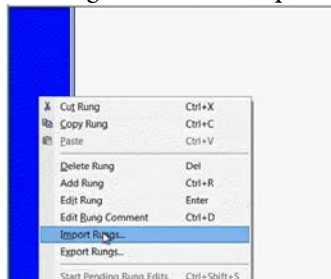


Add the Logic Rung

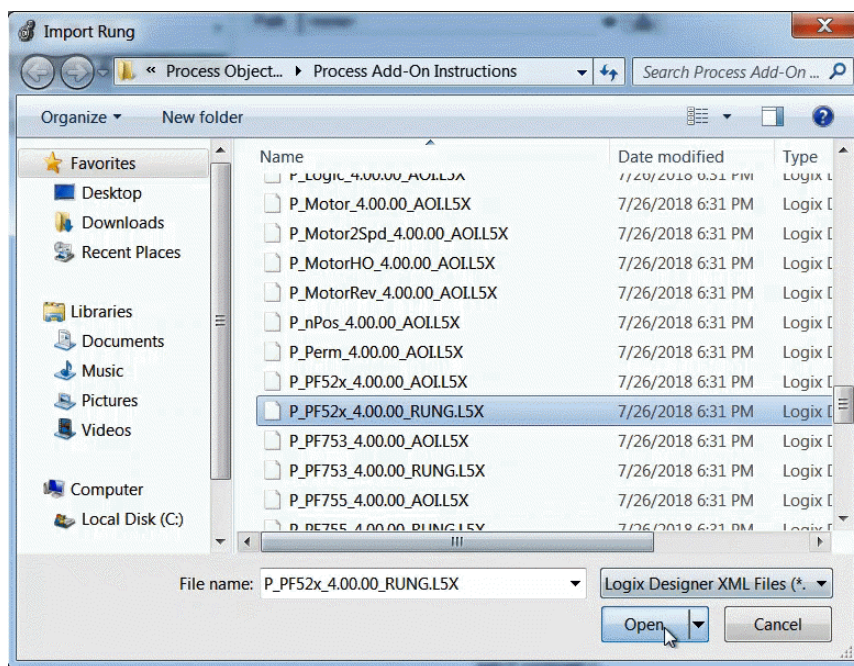
Follow these steps to import a rung into your project.

1. On the Controller Organizer, under Tasks, click in front of Main Task.
2. Double-click Main_Routine to open this ladder logic routine.

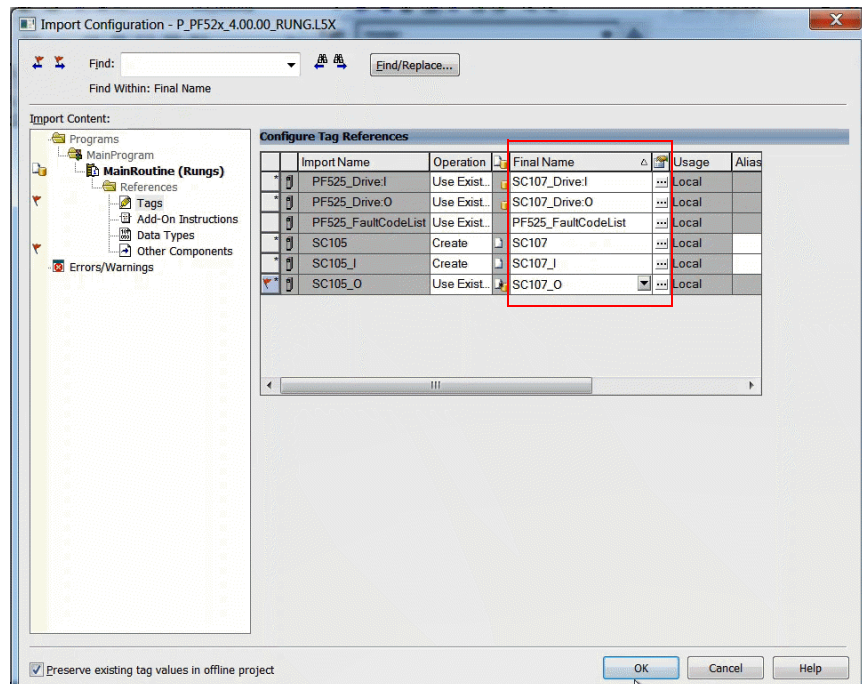
3. Right-click one of the rungs and choose Import Rungs.



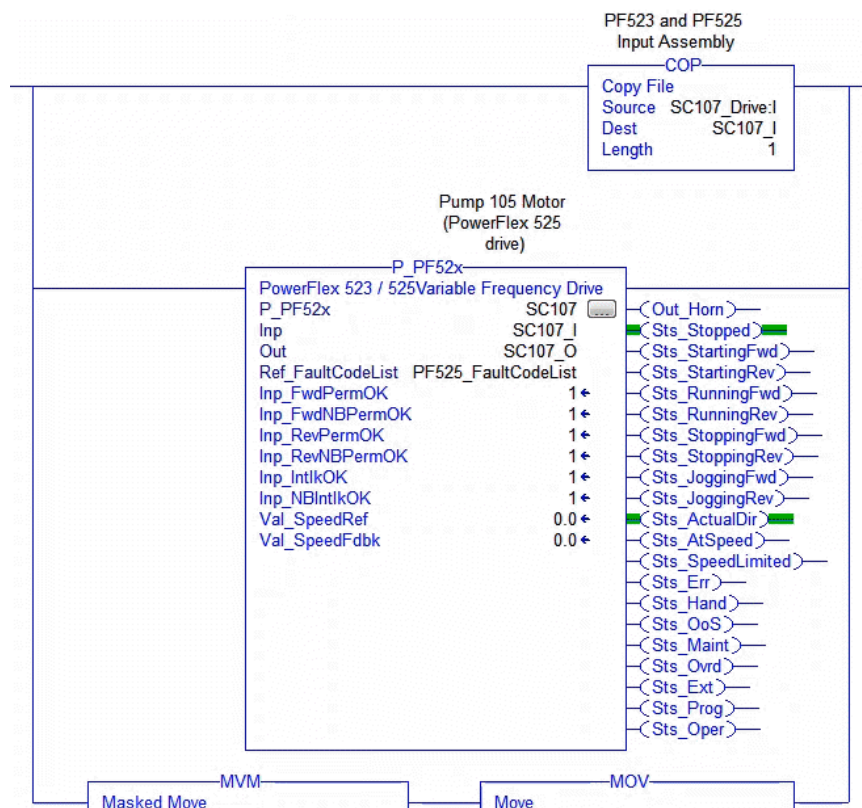
4. On the Import Rungs dialog box, select the device RUNG instruction and click Open.



5. During the import process, you can name the tags for the routine in the Import Configuration dialog box. In the Import Content tree, click Tags and type the names of the variables that match your process and the drive name in the Final Name column. Click OK when finished.



Your ladder logic routine now looks like the example. Observe that the tag names and the drive name are automatically placed in the instruction.



PowerFlex 753 Drive (P_PF753)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_PF753 (PowerFlex 753 Drive) object is used to operate one variable-speed motor by using a PowerFlex 753 AC variable-frequency drive in a variety of modes, and monitoring for fault conditions.

IMPORTANT If your PowerFlex 753 drive uses the 20-750-ENETR Ethernet interface, use the P_PF755 instruction instead. The PowerFlex 753 drive with the 20-750-ENETR interface uses the same interface data structures as the PowerFlex 755 drive, giving you more data.

If you are using a drive other than the PowerFlex 523, 525, 753, 755, 6000, or 7000 drive, use the P_VSD (generic variable-speed drive) instruction instead.

Functional Description

The P_PF753 instruction provides the following capabilities:

- Control of the drive through the standard P_CmdSrc Add-On Instruction and modes.
- Ability to start and stop the drive and motor, control the drive speed (via speed reference), and monitor the drive run status and speed feedback to verify that the drive is running or stopped. Provides alarms and drive shutdown for Fail to Start and Fail to Stop if the feedback does not follow the commanded state within a configured amount of time.
- Reading from the drive, the instruction displays drive faults, drive alarms, conditions that inhibit starting the drive, drive predictive maintenance data, and general drive status data.
- Ability to read a fault code from the drive and provide descriptive text of fault codes.
- Indication of Accelerating, Decelerating, At Speed, Warning, or Alarm status as received from the drive.
- Optional capability to support reversing drives, with commands for forward and reverse rotation, and display of actual rotation direction.
- Input and alarm for a drive fault condition and an output to send a drive fault reset to the drive. Provides a configurable time to pulse the drive fault reset output when a reset command is received.

- Permissives (bypassable and non-bypassable), which are conditions that let a drive start, and Interlocks (bypassable and non-bypassable), which are conditions that stop the drive and prevent starting. Provides an alarm when an Interlock stops the drive. Provides maintenance personnel with the capability to bypass the bypassable Permissives and Interlocks.
- Capability for maintenance personnel to take the drive out of service.

IMPORTANT This capability is not a substitute for hard lockout/tagout (LOTO) procedures.

- Monitor an I/O fault input, and alarm on an I/O fault. The I/O fault condition can optionally de-energize the outputs to the drive, requiring a reset.
- In Override command source, provides an override state input that determines if the override is to run or stop the drive (default = stop), and, if the drive is to run, an override speed reference and direction.
- Provides simulation capability. Outputs to the drive are kept de-energized, but the object can be manipulated as if a working drive were present, including a basic ramp-up of speed feedback value on starting and ramp-down on stopping. The simulated ramp-up-to-speed time is configurable. This capability is often used for activities such as system testing and operator training.

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_PF753_4.10.00_RUNG.L5X rung import must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

InOut Structure

InOut parameters are used to link the Add-On Instruction to external tags that contain necessary data for the instruction to operate. These external tags must be of the data type shown.

Tag Name	Data Type	Description
Inp	P_PF753_Inp	Common part of PowerFlex 753 input assembly.
Out	P_PF753_Out	Common part of PowerFlex 753 output assembly.
Ref_GetFaultMSG	MESSAGE	Control structure for the MSG to get fault data from the drive.
Ref_GetFaultDest	P_PFEEmb_FltAlmRec	Destination for the fault data retrieved from the drive.

IMPORTANT The preceding user-defined data types (UDTs) are included in the RUNG import that brings in the P_PF753 Add-On Instruction.

Setup of the MSG Instructions

The Ref_GetFaultMSG and Ref_GetFaultDest In/Out parameters are tags that need to be created. Create the tags as follows:

- Ref_GetFaultMSG (control structure for the MSG to get fault data from the drive)
 - Name the tag “<drive_tag>_Ref_GetFaultMSG”
 - Data Type: MESSAGE
- Ref_GetFaultDest (destination for the fault data retrieved from the drive)
 - Name the tag “<drive_tag>_Ref_GetFaultDest”
 - Data Type: P_PFEEmb_FltAlmRec

The MSG tag is set up as follows:

- Message Type: CIP™ Generic
- Service Type: Get Attribute Single (code 16#0e)
- Class: 16#97
- Instance: 1
- Attribute: 0
- Destination: The tag you created for the Ref_GetFaultDest tag.
- Path (Communication tab): The drive (select from I/O tree)

Figure 2 - 753 MSG Configuration

Message Configuration - MyP_PF753_GetFaultMsg

Configuration Communication Tag

Message Type: CIP Generic

Service Type: Get Attribute Single Source Element: (empty)

Service Code: e (Hex) Class: 97 (Hex) Source Length: 0 (Bytes)

Instance: 1 Attribute: 0 (Hex) Destination: MyP_PF753_GetFat

New Tag...

☐ Enable
 ☐ Enable Waiting
 ☐ Start
 ☐ Done Done Length: 0

☐ Error Code
 Extended Error Code:
 ☐ Timed Out

Error Path:

Error Text:

OK Cancel Apply Help

Operations

This section describes the primary operations for this Add-On Instruction.

IMPORTANT See [Appendix A](#) for Command Sources and Simulation types.

Alarms

This instruction uses the following alarms, which are implemented by using embedded P_Alarm and P_Gate Add-On Instructions.


Alarm Name	P_Alarm Name	P-Gate Name	Description
Drive Fault	DriveFault	None	Raised when the drive detects a fault and sets its Faulted status bit. Check the Fault Code and description to determine the cause. Issuing a Reset of this object causes a Clear Fault command to be sent to the drive in an attempt to clear the fault.
Fail to Start	FailToStart	None	Raised when the drive has and is using run feedback, an attempt is made to start the drive, and the run feedback does not indicate that the drive is running within the configured time. If Fail to Start is configured as a shed fault, the drive is stopped and a reset is required to start the drive.

Alarm Name	P_Alarm Name	P-Gate Name	Description
Fail to Stop	FailToStop	None	Raised when the drive has and is using run feedback, an attempt is made to stop the drive, and the run feedback does not indicate that the drive stopped within the configured time.
Interlock Trip	IntlkTrip	None	Raised when the drive is running and an interlock 'not OK' condition causes the drive to stop. If interlocks are not bypassed, a bypassable interlock or a non-bypassable interlock 'not OK' condition initiates an interlock trip. If interlocks are bypassed, only a non-bypassable interlock 'not OK' condition initiates an interlock trip.
I/O Fault	IOFault	None	Raised when the Inp_IOFault input is true. This input is used to indicate to the instruction that a communication failure has occurred for its I/O. If the I/O Fault is configured as a shed fault, the drive is stopped and not permitted to start until reset.

Parameters of the P_Alarm object can be accessed by using the following convention: [P_Alarm Name].[P_Alarm Parameter].

Simulation

When P_PF753 is in simulation, the instruction keeps its outputs de-energized and simulates a working drive.

Set the Inp_Sim parameter in the Controller to '1' to enable simulation. When the drive is in simulation, the Simulation or Loopback Test icon  is displayed at the top left of the Operator faceplate.

When in simulation, the object responds to start, stop and jog commands and speed reference changes as if a working drive were being controlled. The configuration parameter Cfg_SimRampT sets the amount of time the simulation takes to ramp from zero speed to maximum and vice versa. The stopped status is asserted when the simulated speed feedback is zero, and running/jogging status is asserted when the simulated speed is greater than zero. There are also configuration bits to determine how speed feedback is calculated based on the speed reference. See the faceplate to see how these configuration bits perform the scaling.

When you have finished in simulation, set the Inp_Sim parameter in the Controller to '0' to return the motor to normal operation.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	Processing for EnableIn False (false rung) is handled the same as if the drive were taken out of service by Command. The drive outputs are de-energized and the drive is shown as Program Out of Service on the HMI. All alarms are cleared.
Powerup (prescan, first scan)	Processing of modes and alarms on Prescan and Powerup is handled by the embedded P_CmdSrc and P_Alarm Add-On Instructions. See their specifications for details. On Powerup, the drive is treated as if it had been Commanded to Stop.
Postscan (SFC Transition)	No SFC Postscan logic is provided.

For more information, see the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#).

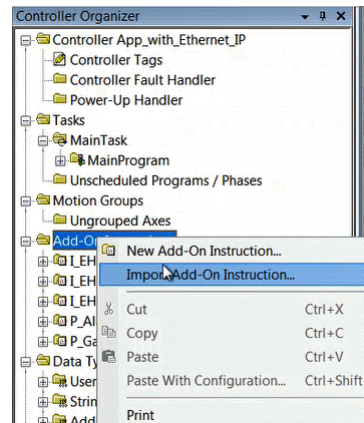
Programming Example

This example will explain how to add the device instruction into your project. Ensure your project is open. For this example we will use P_PF753.

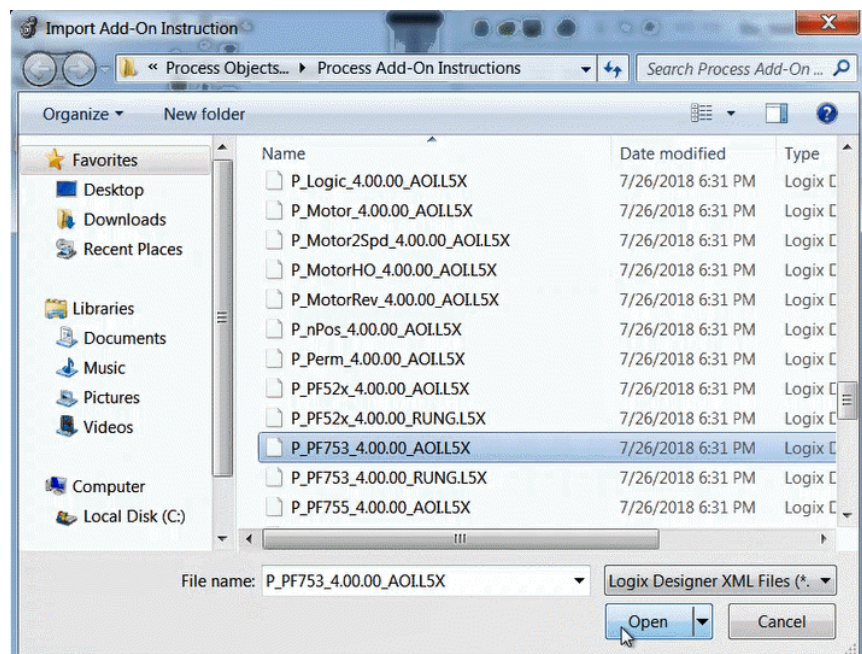
Import Device Add-On Instruction

This procedure imports the definitions for the device Add-On Instruction. It is only necessary to import the definitions once per controller.

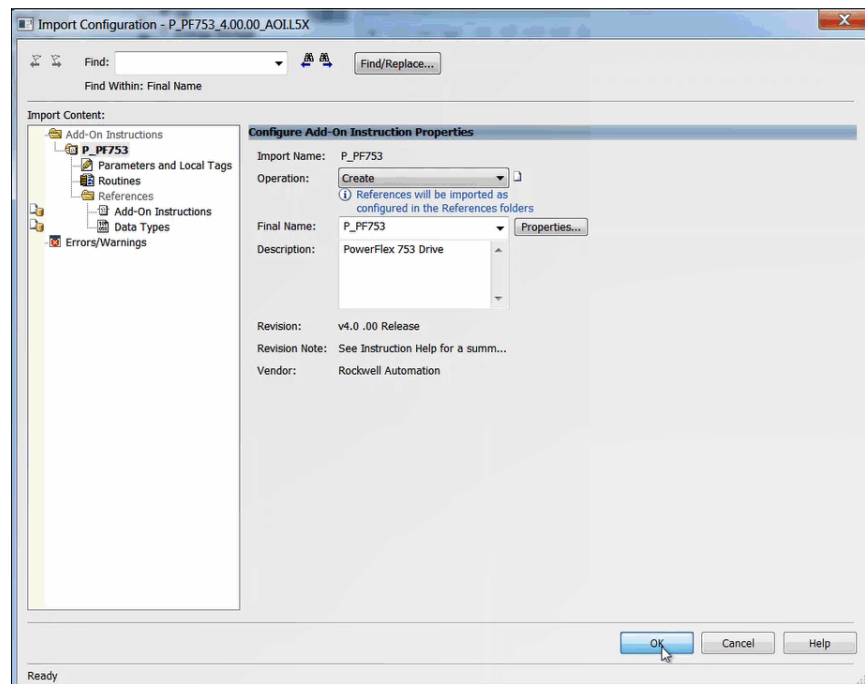
1. In the target Controller Organizer, right-click on Add-On Instructions and choose Import Add-On Instruction.



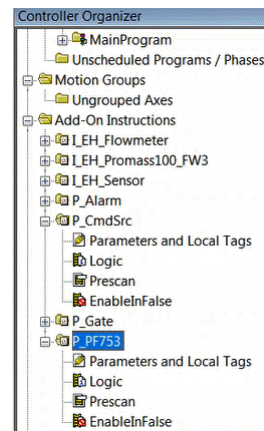
2. Navigate to the folder that contains the device Add-On Instructions, select the appropriate instruction, and click open.



- Click OK in the Import Configuration window.

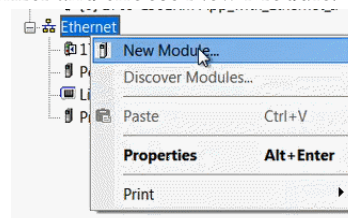


- The Add-On Instruction is then added to the Controller Organizer.

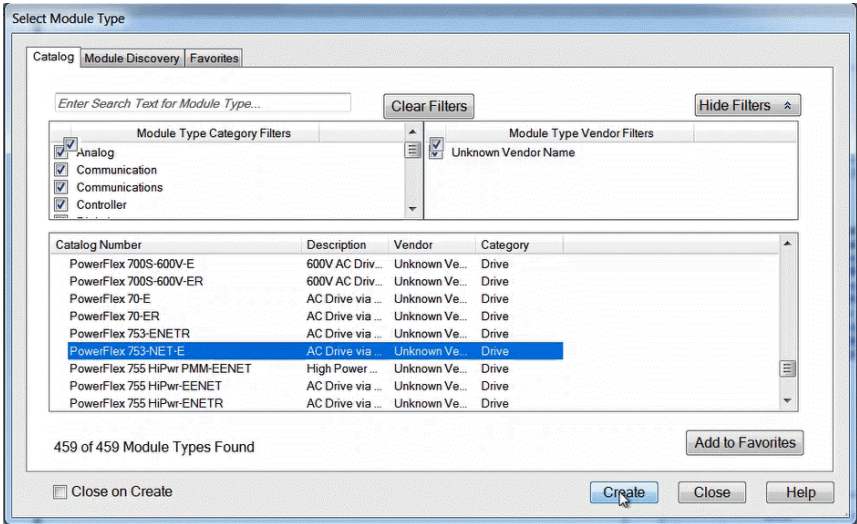


Add the Device to the I/O Tree and Configure the Device

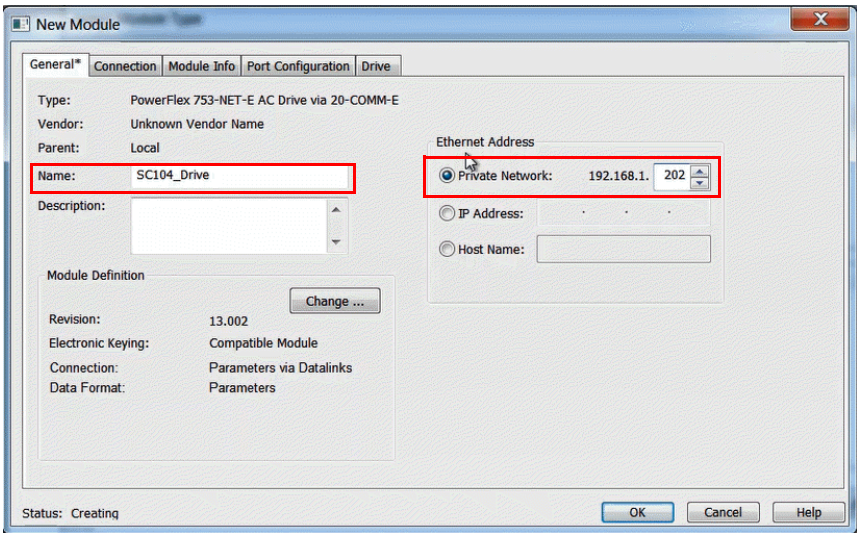
- In your target application, right-click the Ethernet network in the Controller Organizer and choose New Module.



2. Select the module type and click Create.



3. Change the device name and IP address to match the specifications of your project.



4. Click Change in the Module Definition section.

New Module

General* | Connection | Module Info | Port Configuration | Drive

Type: PowerFlex 753-NET-E AC Drive via 20-COMM-E
 Vendor: Unknown Vendor Name
 Parent: Local
 Name: SC104_Drive
 Description:

Module Definition

Revision: 13.002
 Electronic Keying: Compatible Module
 Connection: Parameters via Datalinks
 Data Format: Parameters

Ethernet Address

☒ Private Network: 192.168.1. 202
☐ IP Address:
☐ Host Name:

Status: Creating

OK Cancel Help

5. Change the information for Revision, Electronic Keying, Drive Rating, Rating Options, and Special types to match the specifications of your project.

Module Definition

Revision: 13 2
 Electronic Keying: Compatible Module
 Drive Rating: 200V 4.8 (ND) 4.8 (HD)
 Rating Options: Normal Duty
 Special Types: Compact

Selected Rating: 200V 4.8A
 Selected Catalog: 20F...B4P8

Connection: Parameters via Datalinks
 Data Format: Parameters

Datalink

	Input Data	Output Data
	DriveStatus	LogicCommand
	Feedback	Reference
<input type="checkbox"/> A		<input checked="" type="checkbox"/> Use Network Reference
<input type="checkbox"/> B		
<input type="checkbox"/> C		
<input type="checkbox"/> D		

DANGER: Unexpected, hazardous motion of machinery may occur when improperly using software to configure a drive.

Parameter names selected for the Input and Output Data appear as member names in the drive Module-Defined Data Types and defines necessary Datalink parameters in the RSLogix 5000 project. Actual data transfer between controller and drive is determined by Datalink parameters.

You must download configuration to the drive to ensure that the controller, drive and communication module configurations are consistent with each other.

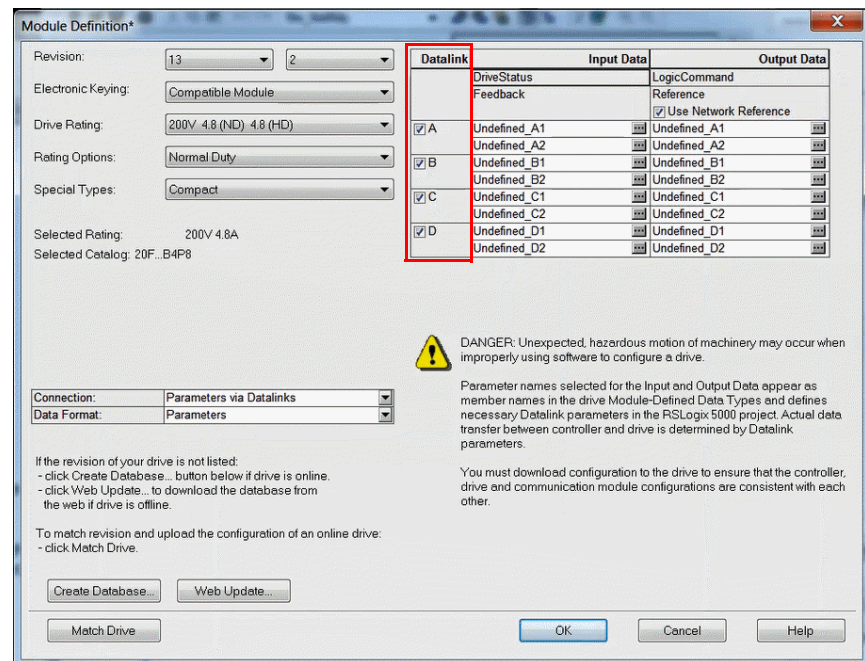
If the revision of your drive is not listed:
 - click Create Database... button below if drive is online.
 - click Web Update... to download the database from the web if drive is offline.

To match revision and upload the configuration of an online drive:
 - click Match Drive.

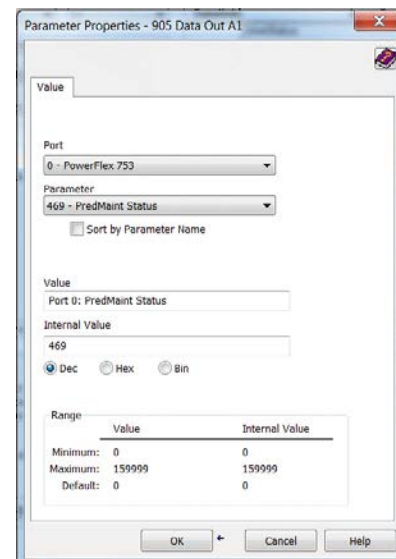
Create Database... Web Update... Match Drive

OK Cancel Help

6. Check the boxes in the Datalink column to add the datalinks.



7. In the Input Data column, click Browse (...). The Parameter Properties dialog box appears.



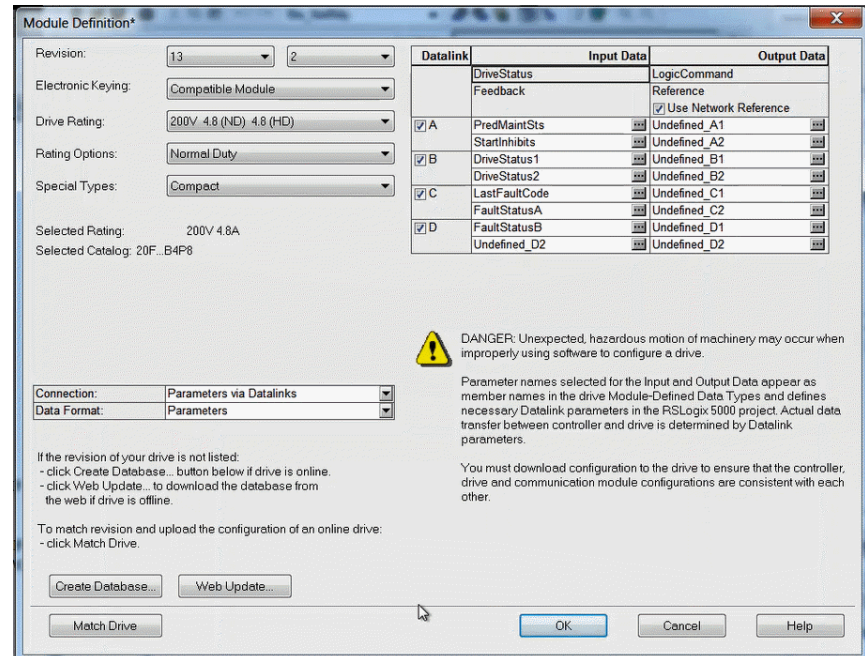
8. From the pull-down menu, choose the port and parameter for each Datalink and click OK. You will return to the Module Definition dialog box after each Datalink.

9. Repeat steps 7 and 8 for each Datalink

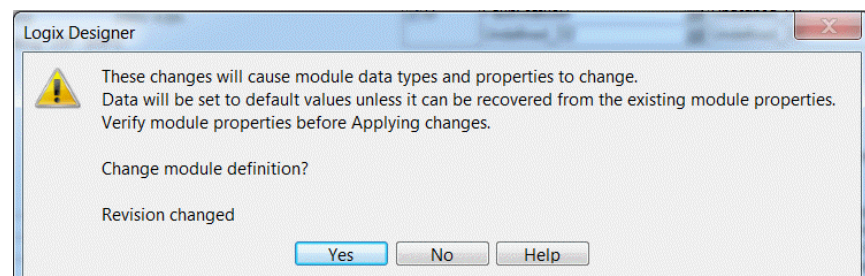
These are the required DataLinks to add to your project:

- Predictive Maintenance Status (Port 0, Parameter 933)
- Start Inhibits (Port 0, Parameter 933)
- Drive Status 1 (Port 0, Parameter 935)

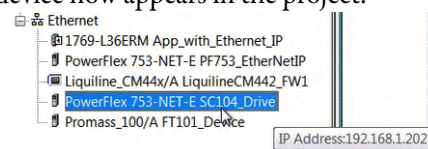
- Drive Status 2 (Port 0, Parameter 936)
 - Last Fault Code (Port 0, Parameter 951)
 - Fault Status A (Port 0, Parameter 952)
 - Fault Status B (Port 0, Parameter 953)
10. Once all Datalinks are provisioned click OK in the Module Definition dialog box.



11. Click YES in the confirmation dialog box to accept the changes to the datalinks.



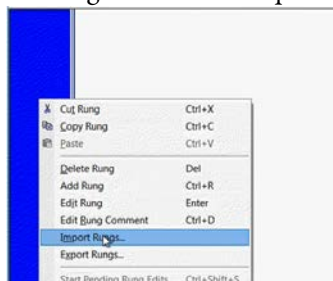
12. The selected device now appears in the project.



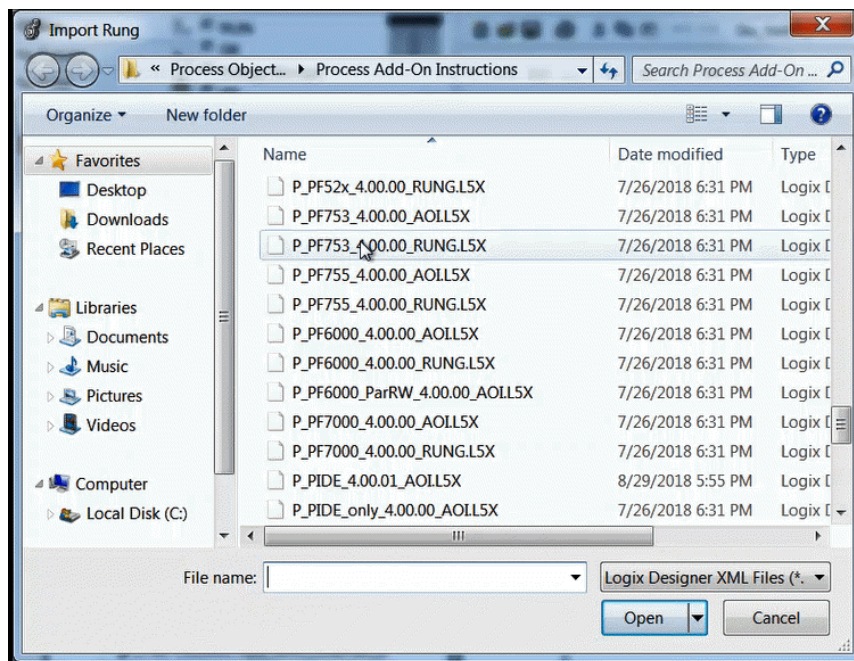
Add the Logic Rung

Follow these steps to import a rung to your project.

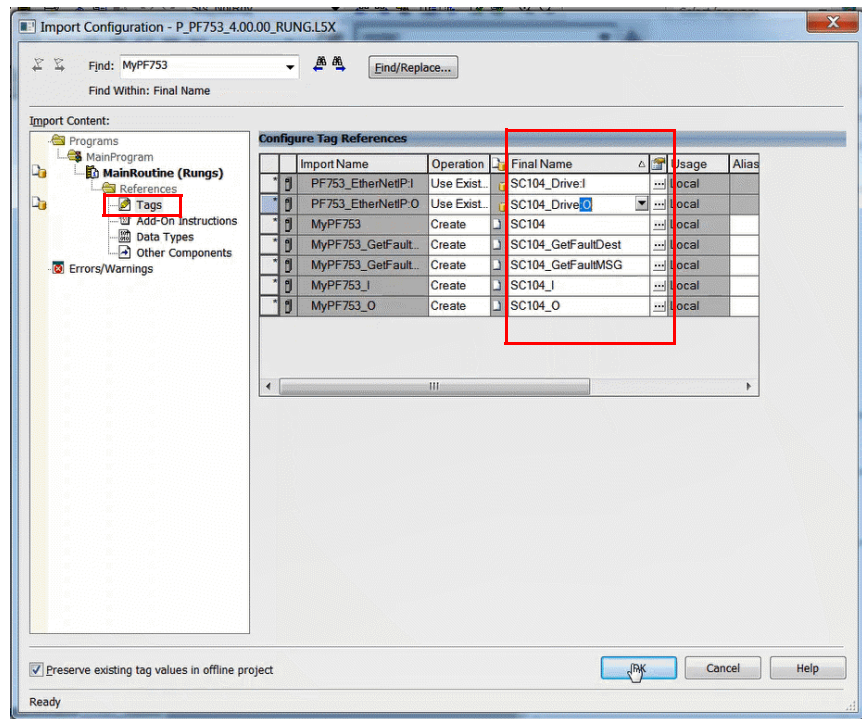
1. On the Controller Organizer, under Tasks, click the plus sign (+) in front of Main Task.
2. Double-click Main_Routine to open this ladder logic routine.
3. Right-click one of the rungs and choose Import Rungs.



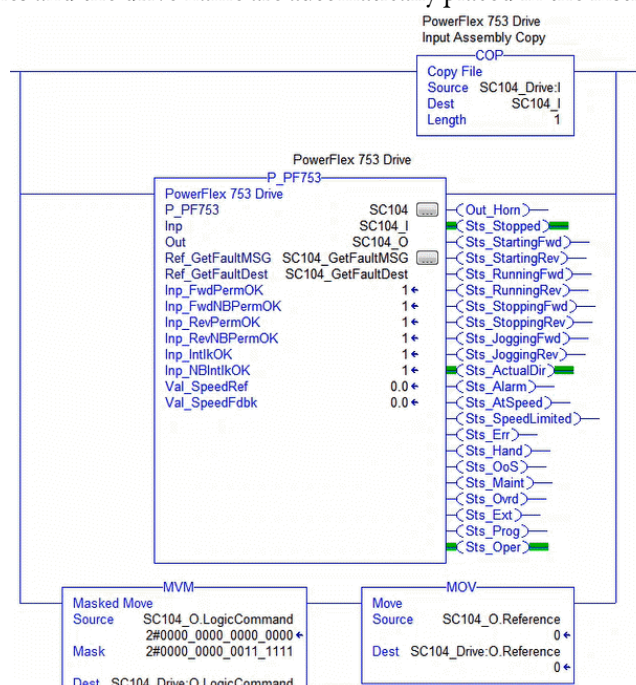
4. On the Import Rungs dialog box, select the device RUNG instruction and click OK.



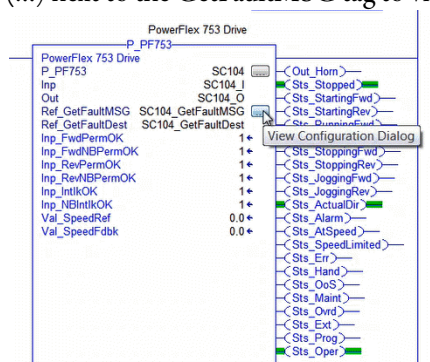
5. During the import process, you can name the tags for the routine in the Import Configuration dialog box. In the Import Content tree, click Tags and type the names of the variables that match your process and the drive name in the Final Name column.



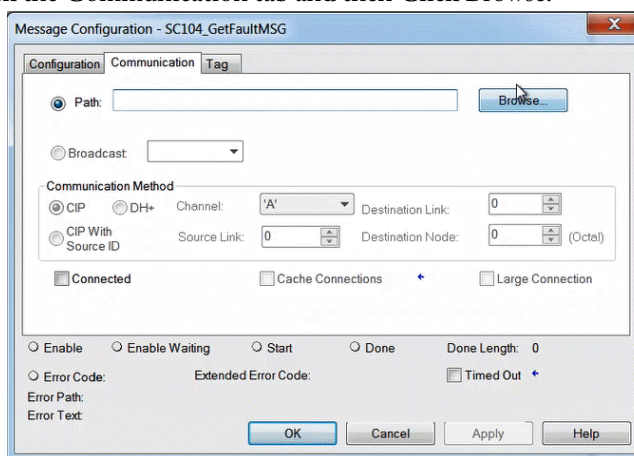
Your ladder logic routine now looks like the example. Observe that the tag names and the drive name are automatically placed in the instruction.



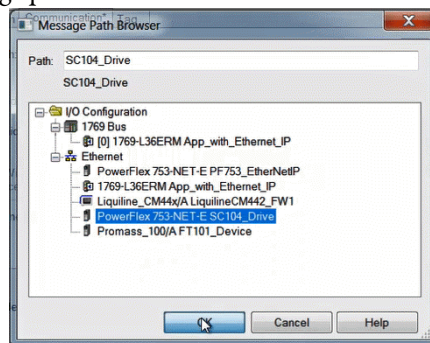
6. Click Browse (...) next to the GetFaultMSG tag to view the configuration.



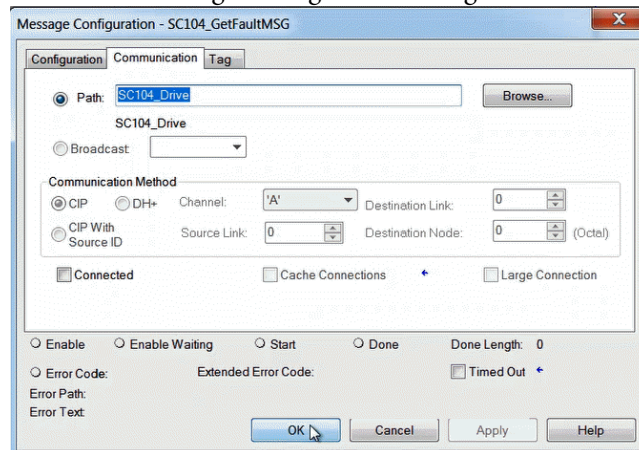
7. Click the Communication tab and then Click Browse.



8. Set the message path to the device and Click OK.



9. Click OK in the Message Configuration dialog box.



PowerFlex 755, PowerFlex 755TL/TR Drives (P_PF755)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_PF755 (PowerFlex 755 drive) object is used to operate one variable-speed motor by using a PowerFlex 755 AC variable frequency drive in a variety of modes and monitoring for fault conditions.

IMPORTANT If you use a drive other than a PowerFlex 755 TL/TR drive or PowerFlex 753 drive with a 20-750-ENETR adapter, use these Add-On Instructions instead:

- P_PF753 for the PowerFlex 753 Drive with 20-COMM-E EtherNet/IP Interface
 - P_PF52x for the PowerFlex 523 or PowerFlex 525 Drive on an EtherNet/IP network
 - P_VSD for third-party drives, drives on other networks, or via hardwired I/O
-

Functional Description

The P_PF755 instruction provides the following capabilities:

- Control of the drive through the standard P_CmdSrc Add-On Instruction.
- Ability to start and stop the drive and motor, control the drive speed (via speed reference), and monitor the drive run status and speed feedback to verify whether the drive is running or stopped. Provides alarms and drive shutdown for Fail to Start and Fail to Stop if the feedback does not follow the commanded state within a configured amount of time.
- Reading from the drive, the instruction displays drive faults, drive alarms, conditions that inhibit starting the drive, drive predictive maintenance data, general drive status data, and a number of operating parameters.
- Ability to read a fault text and descriptive text from the drive.
- Indication of Accelerating, Decelerating, At Speed, Warning, or Alarm status as received from the drive.
- Optional capability to support reversing drives, with commands for forward and reverse rotation, and display of actual rotation direction.
- Input and alarm for a drive fault condition and an output to send a drive fault reset to the drive. Provide a configurable time to pulse the drive fault reset output when a reset command is received.

- Permissives (bypassable and non-bypassable) that are conditions that enable a drive start and Interlocks (bypassable and non-bypassable) that are conditions that stop the drive and prevent starting. Provide an alarm when an Interlock stops the drive. Provide maintenance personnel with the capability to bypass the bypassable Permissives and Interlocks.
- Capability for maintenance personnel to take the drive out of service.

IMPORTANT This capability is not a substitute for hard lockout/tagout (LOTO) procedures.

- Monitor an I/O fault input and alarm on an I/O fault. The I/O fault condition can optionally de-energize the outputs to the drive, requiring a reset.
- In Override command source, provide an override state input that determines if the override is to run or stop the drive (default = stop), and, if the drive is to run, an override speed reference and direction.
- The instruction provides simulation capability. Outputs to the drive are kept de-energized, but the object can be manipulated as if a working drive were present, including a basic ramp-up of speed feedback value on starting and ramp-down on stopping. The simulated ramp-up-to-speed time is configurable. This capability is often used for activities such as system testing and operator training.

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

- The **P_PF755_4.10.00_RUNG.L5X** rung import must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Required Drive Configuration

IMPORTANT Be certain to set up the drive Datalinks as follows!
 'User Choice' Datalinks are not used by this Add-On Instruction and
 may be left unused or set up for your application.

In order for the faceplate to properly display the status of the drive, you must set the Cfg_PF755T parameter to match the drive type. When this bit is 0, the faceplate pages show the status for the basic PowerFlex 755 drive. When this bit is 1, the faceplate shows the status for the PowerFlex 755TL or PowerFlex 755TR drive.

<input type="checkbox"/>	Cfg_PF755T	BOOL	<input type="checkbox"/>	0 Decimal	<input type="checkbox"/>	1=Drive is a PowerFlex 755TL or -TR (TotalFORCE); 0=PowerFlex 755	Read/Write
--------------------------	------------	------	--------------------------	-----------	--------------------------	---	------------

INPUT ASSEMBLY	PowerFlex 755	PowerFlex 755TL, PowerFlex 755TR (TotalFORCE® Drives)
Drive Status	(standard)	(standard)
Feedback	(standard)	(standard)
1. Torque Current Feedback	(755: Par 5)	(755TL/TR: Port 10 Par 8)
2. Output Current	(755: Par 7)	(755TL/TR: Port 10 Par 3)
3. Output Power	(755: Par 9)	(755TL/TR: Port 10 Par 4)
4. Elapsed MWH	(755: Par 13)	(755TL/TR: Port 0 Par 5)
5. Elapsed Run time	(755: Par 15)	(755TL/TR: Port 0 Par 7)
6. Speed Units	(755: Par 300)	(755TL/TR: Port 0 Par 47)
7. Predictive Maintenance Status	(755: Par 469)	(755TL/TR: Port 0 Par 500)
8. Start Inhibits	(755: Par 933)	(755TL/TR: Port 0 Par 603)
9. Drive Status 2	(755: Par 936)	(755TL/TR: Port 10 Par 355)
10. Drive Overload Count	(755: Par 940)	(755TL/TR: Port 10 Par 207)
11. Drive Temperature (Deg. C)	(755: Par 944)	(755TL/TR: Port 0 Par 25)
12. Last Fault Code	(755: Par 951)	(755TL/TR: Port 0 Par 610)
13. Fault Status A	(755: Par 952)	(755TL/TR: Port 10 Par 461)
14. Fault Status B	(755: Par 953)	(755TL/TR: Port 10 Par 462)
15. User choice #1	—	—
16. User choice #2	—	—

InOut Structure

InOut parameters are used to link the Add-On Instruction to external tags that contain necessary data for the instruction to operate. These external tags must be of the data type shown.

Tag Name	Data Type	Description
Inp	P_PF755_Inp	Common part of PowerFlex 755 input assembly.
Out	P_PF755_Out	Common part of PowerFlex 755 output assembly.
Ref_GetFaultMSG	MESSAGE	Control structure for the MSG to get fault data from the drive.
Ref_GetFaultDest	P_PFEmb_FltAlmRec	Destination for the fault data retrieved from the drive.

IMPORTANT The preceding user-defined data types (UDTs) are included in the RUNG import that brings in the P_PF755 Add-On Instruction.

Setup of the MSG Instructions

The Ref_GetFaultMSG and Ref_GetFaultDest In/Out parameters are tags that need to be created. Create the tags as follows:

- Ref_GetFaultMSG (control structure for the MSG to get fault data from the drive)
 - Name the tag “<drive_tag>_Ref_GetFaultMSG”
 - Data Type: MESSAGE
- Ref_GetFaultDest (destination for the fault data retrieved from the drive)
 - Name the tag “<drive_tag>_Ref_GetFaultDest”
 - Data Type: P_PFEmb_FltAlmRec

The MSG tag is set up as follows:

- Message Type: CIP™ Generic
- Service Type: Get Attribute Single (code 16#0e)
- Class: 16#97
- Instance: 1
- Attribute: 0
- Destination: The tag you created for the Ref_GetFaultDest tag.
- Path (Communication tab): The drive (select from I/O tree)

Figure 3 - 755 MSG Configuration

Message Configuration - MyP_PF755_GetFaultMSG

Configuration | Communication | Tag

Message Type: CIP Generic

Service Type: Get Attribute Single Source Element:

Service Code: e (Hex) Class: 97 (Hex) Source Length: 0 (Bytes)

Instance: 1 Attribute: 0 (Hex) Destination: _PF755_GetFaultDest

New Tag...

☐ Enable ☐ Enable Waiting ☐ Start ☐ Done Done Length: 0

☐ Error Code: Extended Error Code: ☐ Timed Out

Error Path:

Error Text:

OK Cancel Apply Help

Operations

This section describes the primary operations for this Add-On Instruction.

IMPORTANT See [Appendix A](#) for Command Sources and Simulation types.

Alarms


This instruction uses the following alarms, which are implemented by using embedded P_Alarm and P_Gate Add-On Instructions.

Alarm Name	P_Alarm Name	P-Gate Name	Description
Drive Fault	DriveFault	None	Raised when the drive detects a fault and sets its Faulted status bit. Check the Fault Code and description to determine the cause. Issuing a Reset of this object causes a Clear Fault command to be sent to the drive in an attempt to clear the fault.
Fail to Start	FailToStart	None	Raised when the drive has and is using run feedback, an attempt is made to start the drive, and the run feedback does not indicate that the drive is running within the configured time. If Fail to Start is configured as a shed fault, the drive is stopped and a reset is required to start the drive.
Fail to Stop	FailToStop	None	Raised when the drive has and is using run feedback, an attempt is made to stop the drive, and the run feedback does not indicate that the drive stopped within the configured time.
Interlock Trip	IntlkTrip	None	Raised when the drive is running and an interlock 'not OK' condition causes the drive to stop. If interlocks are not bypassed, a bypassable interlock or a non-bypassable interlock 'not OK' condition initiates an interlock trip. If interlocks are bypassed, only a non-bypassable interlock 'not OK' condition initiates an interlock trip.
I/O Fault	IOFault	None	Raised when the Inp_IOFault input is true. This input is used to indicate to the instruction that a communication failure has occurred for its I/O. If the I/O Fault is configured as a shed fault, the drive is stopped and not permitted to start until reset.

Parameters of the P_Alarm object can be accessed by using the following convention: [P_Alarm Name].[P_Alarm Parameter].

Simulation

When P_PF755 is in simulation, the instruction keeps its outputs de-energized and simulates a working drive.

Set the Inp_Sim parameter in the Controller to '1' to enable simulation. When the drive is in simulation, the Simulation or Loopback Test icon  is displayed at the top left of the Operator faceplate.

When in simulation, the object responds to start, stop and jog commands and speed reference changes as if a working drive were being controlled. The configuration parameter Cfg_SimRampT sets the amount of time the simulation takes to ramp from zero speed to maximum and vice versa. The stopped status is asserted when the simulated speed feedback is zero, and running/jogging status is asserted when the simulated speed is greater than zero. There are also configuration bits to determine how speed feedback is calculated based on the speed reference. See the faceplate to see how these configuration bits perform the scaling.

When you have finished in simulation, set the Inp_Sim parameter in the Controller to '0' to return the motor to normal operation.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	Processing for EnableIn False (false rung) is handled the same as if the drive were taken out of service by Command. The drive outputs are de-energized and the drive is shown as Program Out of Service on the HMI. All alarms are cleared.
Powerup (prescan, first scan)	Processing of modes and alarms on Prescan and Powerup is handled by the embedded P_CmdSrc and P_AlarmAdd-On Instructions. See their specifications for details. On Powerup, the drive is treated as if it had been Commanded to Stop.
Postscan (SFC Transition)	No SFC Postscan logic is provided.

For more information, see the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#).

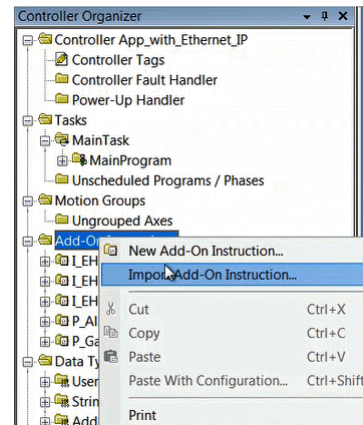
Programming Example

This example will explain how to add the device instruction into your project. Ensure your project is open. For this example we will use P_PF755.

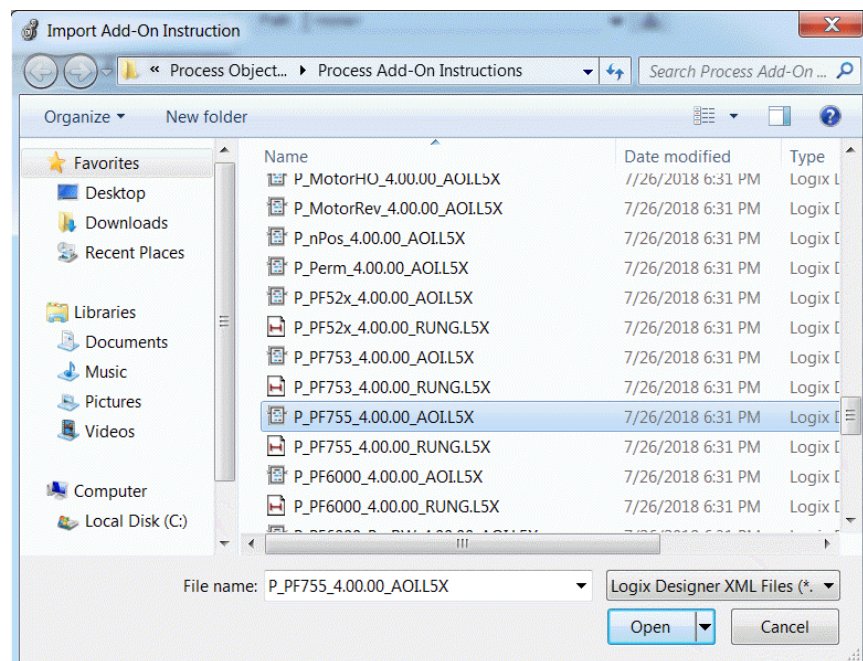
Import Device Add-On Instruction

This procedure imports the definitions for the device Add-On Instruction. It is only necessary to import the definitions once per controller.

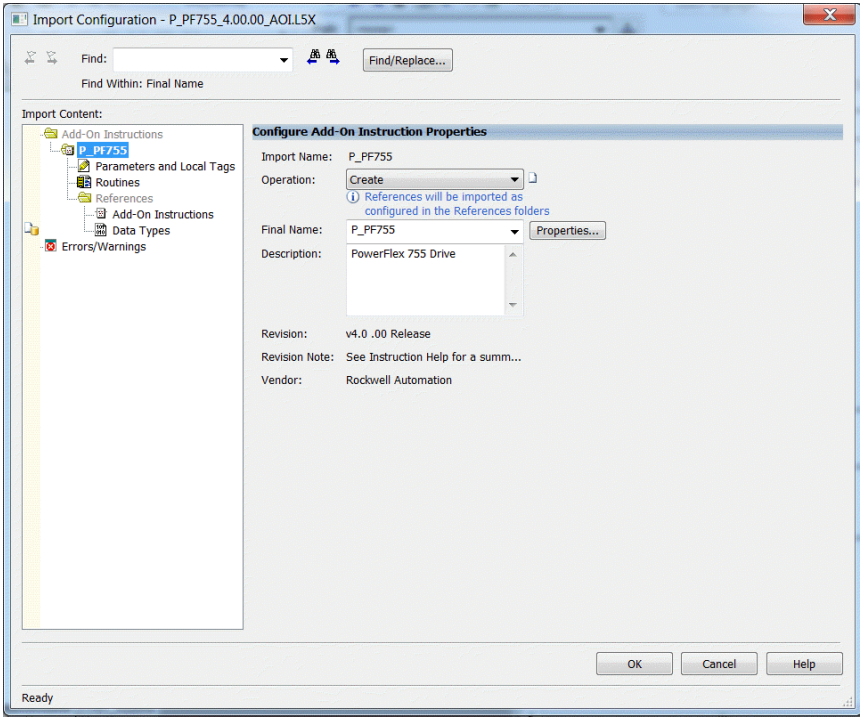
1. In the target Controller Organizer, right-click on Add-On Instructions and choose Import Add-On Instruction.



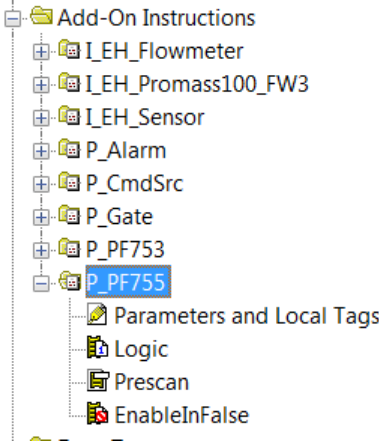
2. Navigate to the folder that contains the device Add-On Instructions, select the appropriate instruction, and click open.



3. Click OK in the Import Configuration window.

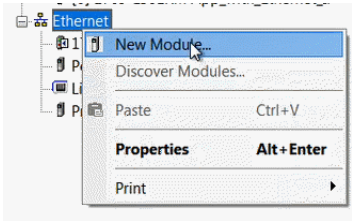


4. The Add-On Instruction is then added to the Controller Organizer.

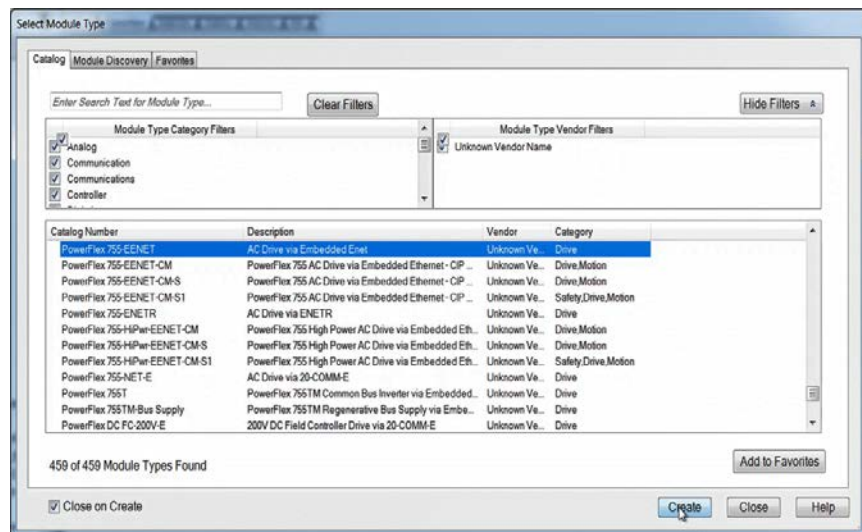


Add the Device to the I/O Tree and Configure the Device

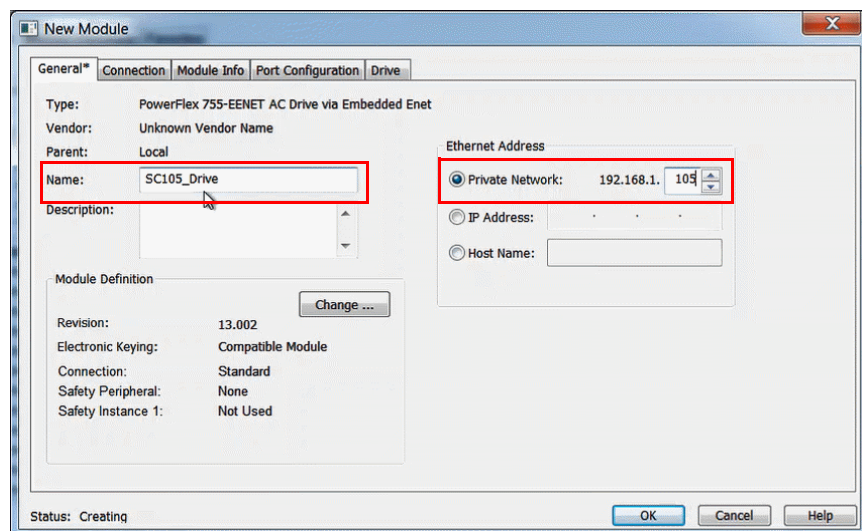
1. In your target application, right-click the Ethernet network in the Controller Organizer and choose New Module.



2. Select the Module Type and click Create.



3. Change the device name and IP address to match the specifications of your project.



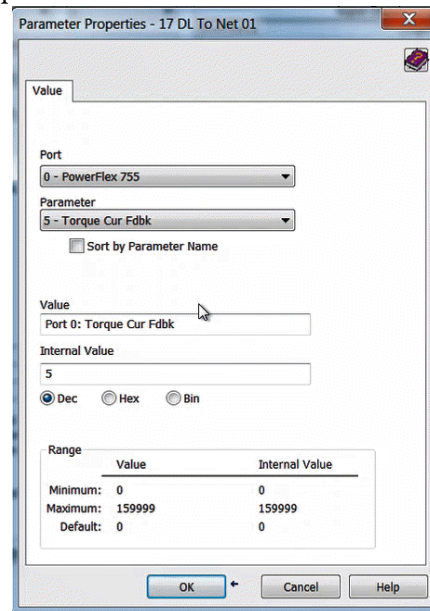
4. Click Change in the Module Definition section.

The 'New Module' dialog box is shown with the 'General' tab selected. The 'Type' is 'PowerFlex 755-EENET AC Drive via Embedded Enet'. The 'Parent' is 'Local'. The 'Name' is 'SC105_Drive'. The 'Description' is empty. The 'Module Definition' section has a 'Change ...' button highlighted with a red box. The 'Revision' is '13.002'. The 'Electronic Keying' is 'Compatible Module'. The 'Connection' is 'Standard'. The 'Safety Peripheral' is 'None'. The 'Safety Instance 1' is 'Not Used'. The 'Ethernet Address' section has 'Private Network' selected with IP '192.168.1.105'. The 'IP Address' and 'Host Name' fields are empty. The 'Status' is 'Creating'. The 'OK', 'Cancel', and 'Help' buttons are at the bottom right.

5. Change the information for Revision, Electronic Keying, Drive Rating, Rating Options, and Special Types to match the specifications of your project.

The 'Module Definition' dialog box is shown. The 'Revision' is '13', 'Electronic Keying' is 'Compatible Module', 'Drive Rating' is '200V 4.8 (ND) 4.8 (HD)', 'Rating Options' is 'Normal Duty', and 'Special Types' is 'Compact'. The 'Selected Rating' is '200V 4.8A' and the 'Selected Catalog' is '20G..B4P8'. The 'Connection' is 'Standard', 'Safety Peripheral' is 'None', and 'Safety Instance 1' is 'Not Used'. The 'Input Data' and 'Output Data' tables are shown. The 'Input Data' table has columns 'DriveStatus' and 'Feedback'. The 'Output Data' table has columns 'LogicCommand' and 'Reference'. The 'Use Network Reference' checkbox is checked. A warning icon and text are present: 'DANGER: Unexpected, hazardous motion of machinery may occur when improperly using software to configure a drive.' Below this, it says: 'Parameter names selected for the Input and Output Data appear as member names in the drive Module-Defined Data Types and defines necessary Datalink parameters in the RSLogix 5000 project. Actual data transfer between controller and drive is determined by Datalink parameters. You must download configuration to the drive to ensure that the controller, drive and communication module configurations are consistent with each other.' At the bottom, there are buttons for 'Create Database...', 'Web Update...', 'Match Drive', 'OK', 'Cancel', and 'Help'.

6. In the Input Data column, click Browse (...). The Parameter Properties dialog box appears.



7. From the pull-down menu, choose the port and parameter for each Datalink and click OK. You will return to the Module Definition dialog box after each Datalink.

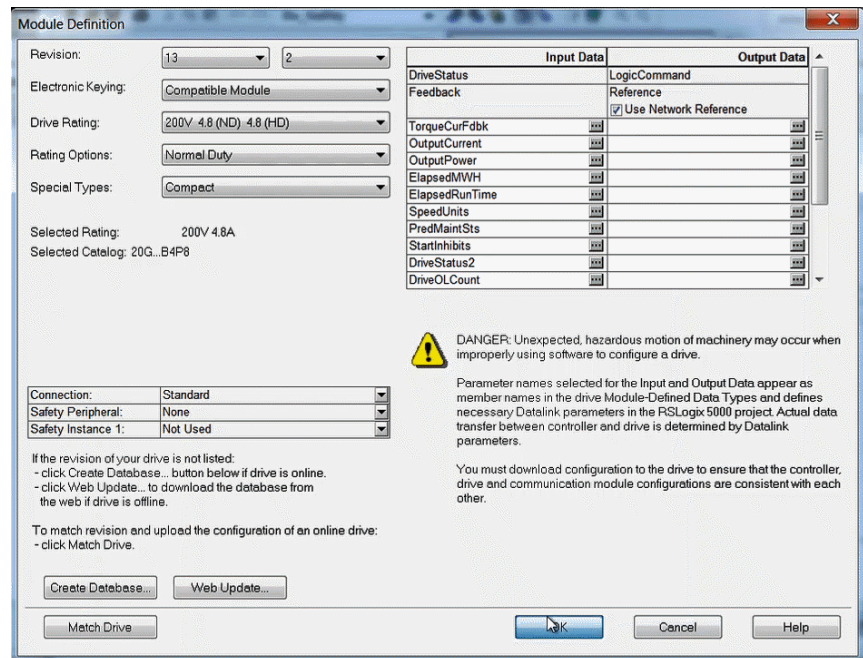
Repeat steps 6 and 7 for each Datalink.

The required DataLinks to add to your project are:

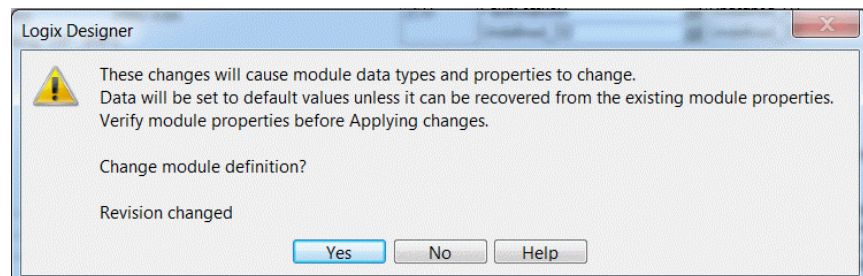
- Torque Current Feedback (Par 5)
- Output Current (Par 7)
- Output Power (Par 9)
- Elapsed MWH (Par 13)
- Elapsed Run time (Par 15)
- Speed Units (Par 300)
- Predictive Maintenance Status (Par 469)
- Start Inhibits (Par 933)
- Drive Status 2 (Par 936)
- Drive Overload Count (Par 940)
- Drive Temperature (C) (Par 944)
- Last Fault Code (Par 951)
- Fault Status A (Par 952)
- Fault Status B (Par 953)

The last two datalinks are not used by this instruction and are available for your application.

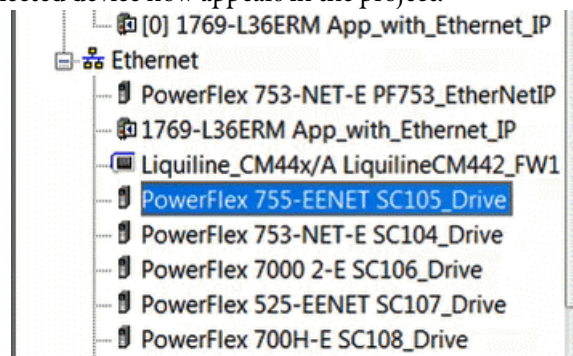
8. Once all Datalinks are provisioned click OK in the Module Definition dialog box.



9. Click YES in the confirmation dialog box to accept the changes to the datalinks.




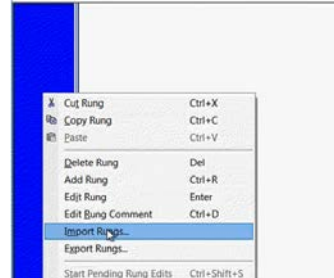
10. The selected device now appears in the project.



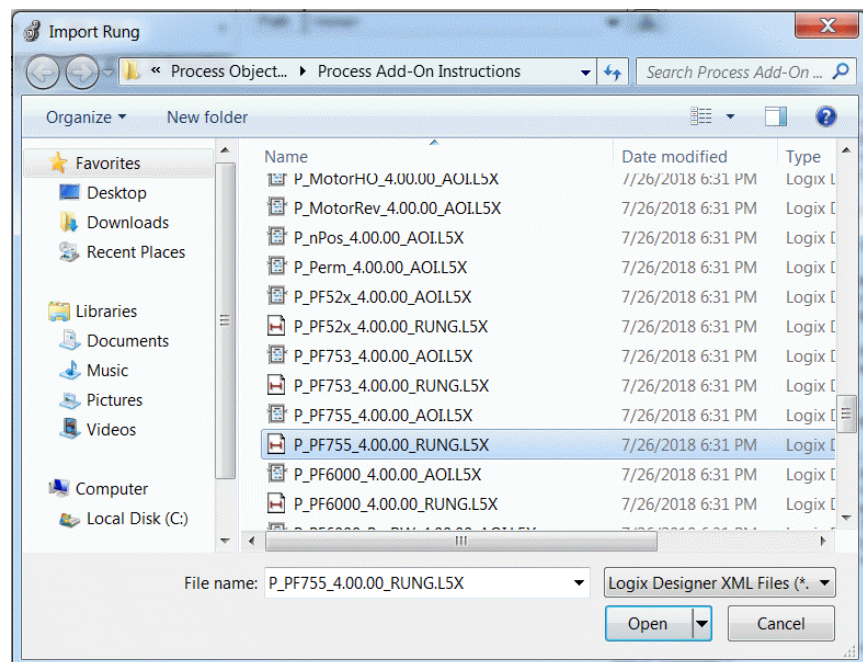
Add the Logic Rung

Follow these steps to import a rung into your project.

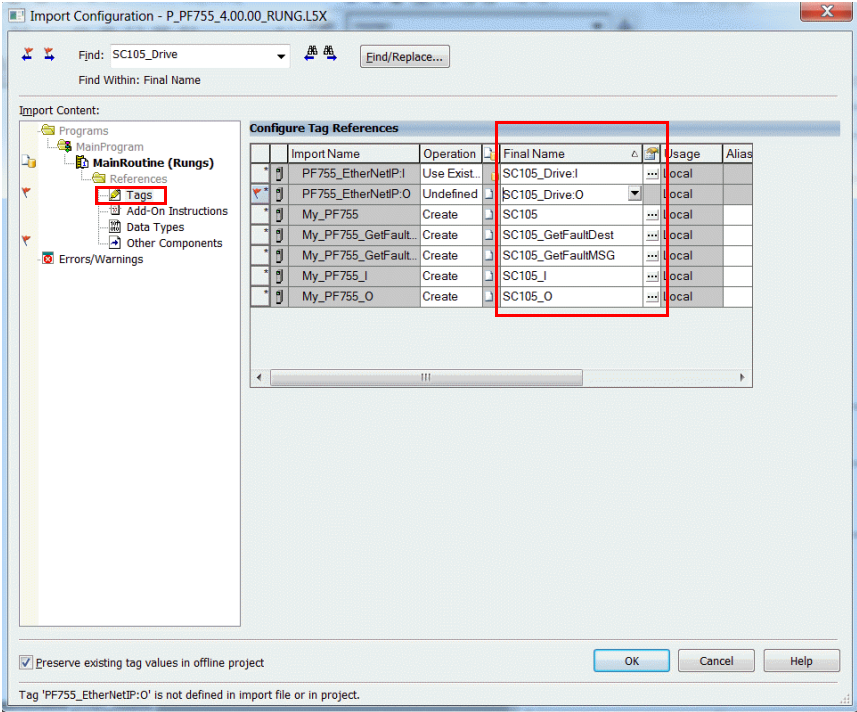
1. On the Controller Organizer, under Tasks, click  in front of Main Task.
2. Double-click Main_Routine to open this ladder logic routine.
3. Right-click one of the rungs and choose Import Rungs.



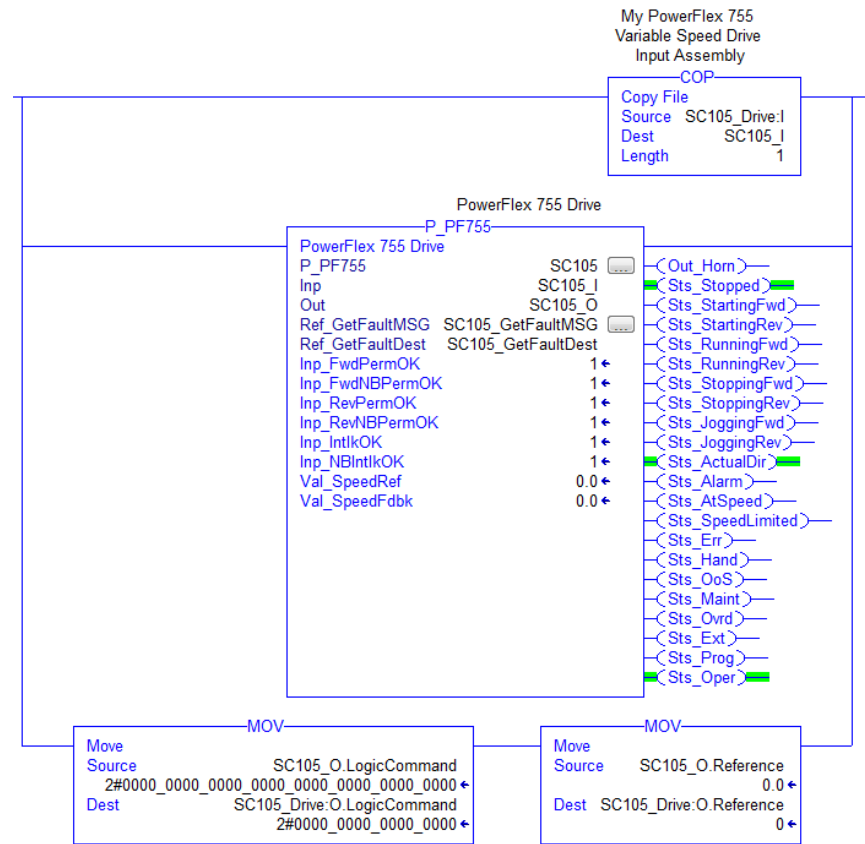
4. On the Import Rungs dialog box, select the device RUNG instruction and click Open.



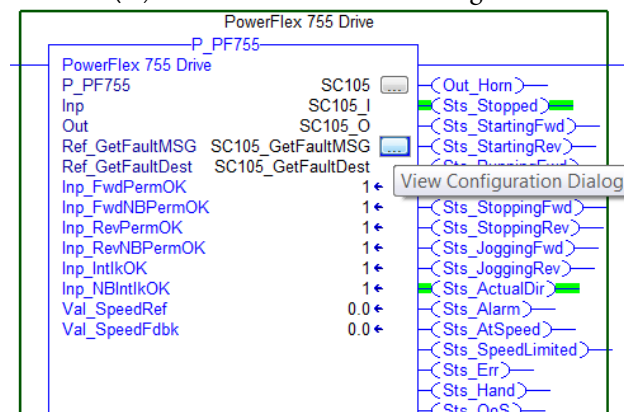
5. During the import process, you can name the tags for the routine in the Import Configuration dialog box. In the Import Content tree, click Tags and type the names of the variables that match your process and the drive name in the Final Name column.



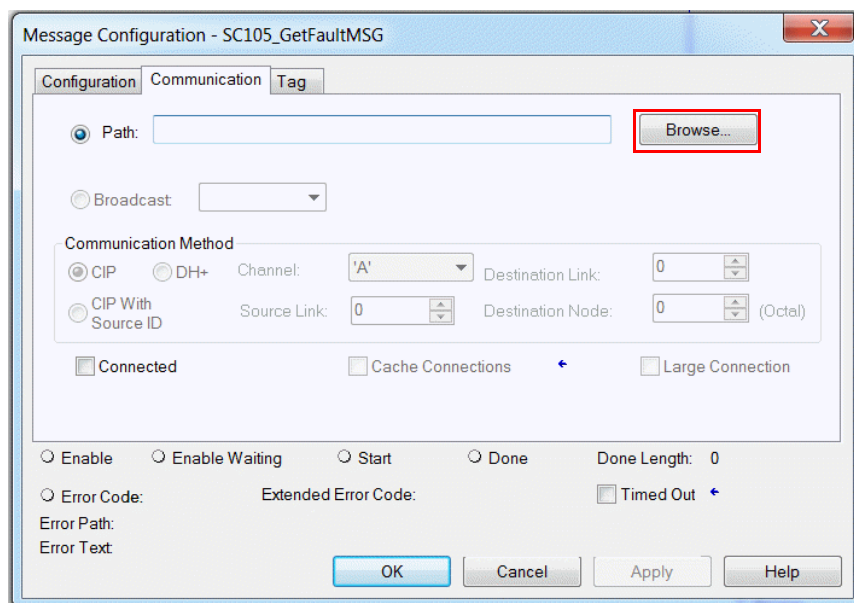
Your ladder logic routine now looks like the example. Observe that the tag names and the drive name are automatically placed in the instruction.



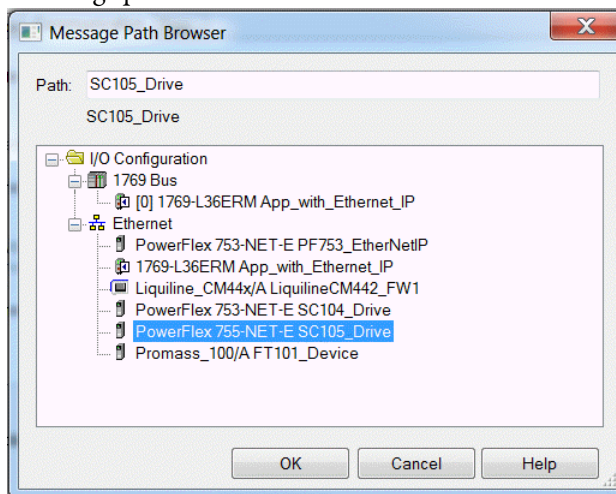
6. Click Browse (...) next to the GetFaultMSG tag to view the configuration.



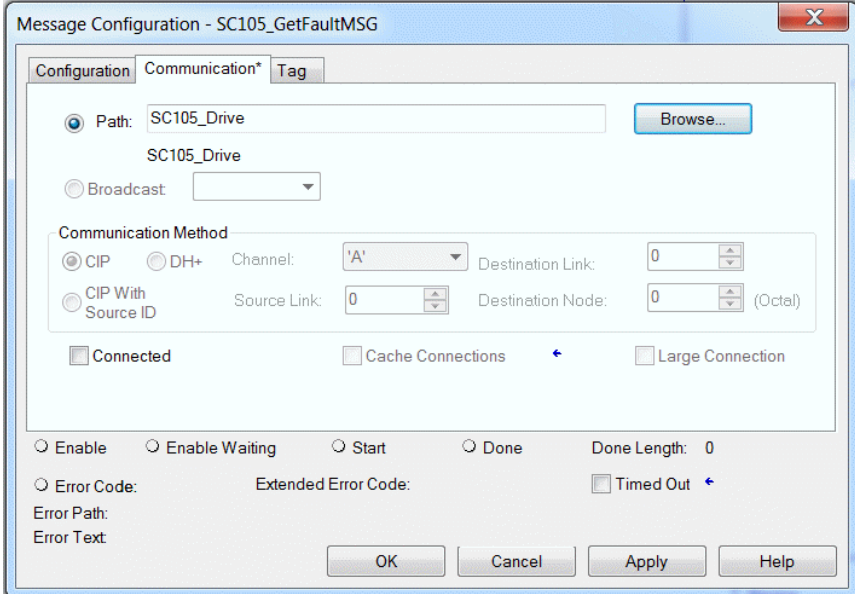
7. Click the Communication tab and then Click Browse.



8. Set the message path to the device and Click OK.



9. Click OK in the Message Configuration dialog box.



The image shows the 'Message Configuration - SC105_GetFaultMSG' dialog box. It has three tabs: 'Configuration', 'Communication*', and 'Tag'. The 'Configuration' tab is active. It contains the following fields and controls:

- Path:** A text box containing 'SC105_Drive' and a 'Browse...' button.
- Broadcast:** A radio button and a dropdown menu.
- Communication Method:**
 - CIP:** A radio button. It is selected. It has a 'Channel' dropdown set to 'A' and a 'Destination Link' spinner set to 0.
 - CIP With Source ID:** A radio button. It is not selected. It has a 'Source Link' spinner set to 0 and a 'Destination Node' spinner set to 0 (labeled '(Octal)').
- Connected:** A checkbox.
- Cache Connections:** A checkbox.
- Large Connection:** A checkbox.
- Enable:** A radio button.
- Enable Waiting:** A radio button.
- Start:** A radio button.
- Done:** A radio button.
- Done Length:** A spinner set to 0.
- Error Code:** A radio button.
- Extended Error Code:** A radio button.
- Timed Out:** A checkbox.
- Error Path:** A text box.
- Error Text:** A text box.
- Buttons:** 'OK', 'Cancel', 'Apply', and 'Help'.

PowerFlex 6000 Drive (P_PF6000)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

IMPORTANT The drive interface is designed to work with the Studio 5000 Logix Designer application, Version 20 and later.

Functional Description

The P_PF6000 instruction provides the following capabilities:

- Control of the drive through the standard P_CmdSrc Add-On Instruction.
- Ability to start and stop the drive and motor, control the drive speed (via speed reference), and monitor the drive run status and speed feedback to verify whether the drive is running or stopped. Provides alarms and drive shutdown for Fail to Start and Fail to Stop if the feedback does not follow the commanded state within a configured amount of time.
- Reading from the drive, the instruction displays drive faults, general drive status data, and a number of operating parameters.
- Ability to read fault data from the drive and provide descriptive text with fault codes.
- Indication of Warning or Alarm status as received from the drive.
- Input and alarm for a drive fault condition and an output to send a drive fault reset to the drive. Provide a configurable time to pulse the drive fault reset output when a reset command is received.
- Permissives (bypassable and non-bypassable) that are conditions that enable a drive start and Interlocks (bypassable and non-bypassable) that are conditions that stop the drive and help prevent starting. Provide an alarm when an Interlock stops the drive. Provide maintenance personnel with the capability to bypass the bypassable Permissives and Interlocks.
- Capability for maintenance personnel to take the drive out of service.

IMPORTANT This capability is not a substitute for hard lockout/tagout (LOTO) procedures.

- Monitor an I/O fault input and a communication watchdog timer, and alarm on an I/O or communication failure. The failure condition can optionally de-energize the outputs to the drive, which requires a reset.
- In Override command source, provide an override state input that determines if the override is to run or stop the drive (default = stop), and, if the drive is to run, an override speed reference and direction.

- The instruction provides simulation capability. Outputs to the drive are kept de-energized, but the object can be manipulated as if a working drive were present, including a basic ramp-up of speed feedback value on starting and ramp-down on stopping. The simulated ramp-up-to-speed time is configurable. This capability is often used for activities such as system testing and operator training.

IMPORTANT If you use a drive other than the PowerFlex 6000 drive, use these Add-On Instructions instead:

- P_PF52x for the PowerFlex 523 or PowerFlex 525 drive on an EtherNet/IP network
- P_PF753 for the PowerFlex drive with 20-COMM-E EtherNet/IP Interface
- P_PF755 for the PowerFlex AC variable-frequency drive
- P_PF7000 for the PowerFlex 7000 medium voltage AC variable-frequency drive with 20-COMM-E EtherNet/IP interface.
- P_VSD for third-party drives, drives on other networks, or via hardwired I/O

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_PF6000_4.10.00_RUNG.L5X rung import must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

InOut Structure

InOut parameters are used to link the Add-On Instruction to external tags that contain necessary data for the instruction to operate. These external tags must be of the data type shown.

Tag Name	Data Type	Description
Inp	P_PF6000_Inp	PowerFlex 6000 input assembly.
Out	P_PF6000_Out	PowerFlex 6000 output assembly.
Ref_FaultCodeList	P_DescList	Tag containing list of fault codes (DINT) and descriptions (STRING_40).

IMPORTANT The user-defined data types (UDTs) and the Array tag containing the list of PowerFlex 6000 fault codes and descriptions are included in the RUNG import that brings in the P_PF6000 Add-On Instruction.

The figure shows the drive fault table tags that are in each template.

Name	Value	Style	Data Type	Description
PF4xx_FaultCodeList	{...}		P_DescList[36]	PowerFlex 4 / 40 / 400 Fault Codes and Descriptions
PF7xx_FaultCodeList	{...}		P_DescList[120]	PowerFlex 70 / 700 / 700H Fault Codes and Descriptions
PF75x_FaultCodeList	{...}		P_DescList[212]	PowerFlex 753 / 755 Fault Codes and Descriptions
PF525_FaultCodeList	{...}		P_DescList[61]	PowerFlex 525 VFD Fault Codes and Descriptions
PF700S_FaultCodeList	{...}		P_DescList[80]	PowerFlex 700S VFD Fault Codes and Descriptions
PF6000_FaultCodeList	{...}		P_DescList[490]	PowerFlex 6000 VFD Fault Codes and Descriptions
PFDC_FaultCodeList	{...}		P_DescList[50]	PowerFlex DC Drive Fault Codes and Descriptions
SMC50_FaultCodeList	{...}		P_DescList[68]	SMC-50 Smart Motor Controller Fault Code List
SMCFlex_FaultCodeList	{...}		P_DescList[48]	SMCFlex Soft Starter Fault Codes and Descriptions

Make sure the tag 'PF6000_FaultCodeList' is entered in the P_PF6000 Ref_FaultCodeList parameter.

Each fault code list provides pre-configured fault codes and descriptions for a given drive family.

Controller Tags - RSL5k_20Emu_ProcessObjects_4_00_02(controller)

Scope: RSL5k_20Emu_1 Show: All Tags

Name	Value	Style	Data Type
PF6000_FaultCodeList	{...}		P_DescList[490]
PF6000_FaultCodeList[0]	{...}		P_DescList
PF6000_FaultCodeList[0].Code	0	Decimal	DINT
PF6000_FaultCodeList[0].Desc	'Check drive manual for this fault code'		STRING_40
PF6000_FaultCodeList[1]	{...}		P_DescList
PF6000_FaultCodeList[1].Code	-1	Decimal	DINT
PF6000_FaultCodeList[1].Desc	'No more fault information found'		STRING_40
PF6000_FaultCodeList[2]	{...}		P_DescList
PF6000_FaultCodeList[2].Code	1	Decimal	DINT
PF6000_FaultCodeList[2].Desc	'IGBT Over Current In Power Cell #1'		STRING_40
PF6000_FaultCodeList[3]	{...}		P_DescList
PF6000_FaultCodeList[3].Code	2	Decimal	DINT
PF6000_FaultCodeList[3].Desc	'Capacitor Abnormal In Power Cell #1'		STRING_40
PF6000_FaultCodeList[4]	{...}		P_DescList
PF6000_FaultCodeList[4].Code	3	Decimal	DINT
PF6000_FaultCodeList[4].Desc	'Input Over Voltage In Power Cell #1'		STRING_40
PF6000_FaultCodeList[5]	{...}		P_DescList
PF6000_FaultCodeList[5].Code	4	Decimal	DINT
PF6000_FaultCodeList[5].Desc	'DCBus Undervolt In Power Cell #1 Warn'		STRING_40
PF6000_FaultCodeList[6]	{...}		P_DescList
PF6000_FaultCodeList[6].Code	5	Decimal	DINT
PF6000_FaultCodeList[6].Desc	'Communication Error In Power Cell #1'		STRING_40
PF6000_FaultCodeList[7]	{...}		P_DescList
PF6000_FaultCodeList[7].Code	6	Decimal	DINT
PF6000_FaultCodeList[7].Desc	'No PWM Pulse For IGBT In Power cell #1'		STRING_40
PF6000_FaultCodeList[8]	{...}		P_DescList
PF6000_FaultCodeList[8].Code	9	Decimal	DINT
PF6000_FaultCodeList[8].Desc	'IGBT Failed To Turn On In Power Cell #1'		STRING_40

For a list of fault codes, refer to the PowerFlex 6000 Medium Voltage Variable-frequency drive Firmware, Parameters, and Troubleshooting Manual, publication [6000-TD004](#).

Operations

This section describes the primary operations for this Add-On Instruction.

Alarms


This instruction uses the following alarms, which are implemented by using embedded P_Alarm Add-On Instructions.

Alarm Name	P_Alarm Name	P-Gate Name	Description
Drive Fault	DriveFault	None	Raised when the drive detects a fault and sets its Faulted status bit. Check the Fault Code and description to determine the cause. Issuing a Reset of this object causes a Clear Fault command to be sent to the drive in an attempt to clear the fault.
Fail to Start	FailToStart	None	Raised when the drive has and is using run feedback, an attempt is made to start the drive, and the run feedback does not indicate that the drive is running within the configured time. If Fail to Start is configured as a shed fault, the drive is stopped and a reset is required to start the drive.
Fail to Stop	FailToStop	None	Raised when the drive has and is using run feedback, an attempt is made to stop the drive, and the run feedback does not indicate that the drive stopped within the configured time.
Interlock Trip	IntlkTrip	None	Raised when the drive is running and an interlock 'not OK' condition causes the drive to stop. If interlocks are not bypassed, a bypassable interlock or a non-bypassable interlock 'not OK' condition initiates an interlock trip. If interlocks are bypassed, only a non-bypassable interlock 'not OK' condition initiates an interlock trip.
I/O Fault	IOFault	None	Raised when the Inp_IOFault input is true. This input is used to indicate to the instruction that a communication failure has occurred for its I/O. If the I/O fault is configured as a shed fault, the drive is stopped and not permitted to start until reset.

Parameters of the P_Alarm object can be accessed by using the following convention: [P_Alarm Name].[P_Alarm Parameter].

Simulation

When P_PF6000 is in simulation, the instruction keeps its outputs de-energized and simulates a working drive.

Set the Inp_Sim parameter in the Controller to '1' to enable simulation. When the drive is in simulation, the Simulation or Loopback Test icon  is displayed at the top left of the Operator faceplate.

When in simulation, the object responds to start, stop and jog commands and speed reference changes as if a working drive were being controlled. The configuration parameter Cfg_SimRampT sets the amount of time the simulation takes to ramp from zero speed to maximum and vice versa. The stopped status is asserted when the simulated speed feedback is zero, and running/jogging status is asserted when the simulated speed is greater than zero. There are also configuration bits to determine how speed feedback is calculated based on the speed reference. See the faceplate to see how these configuration bits perform the scaling.

When you have finished in simulation, set the Inp_Sim parameter in the Controller to '0' to return the motor to normal operation.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	Processing for EnableIn False (false rung) is handled the same as if the drive were taken out of service by Command. The drive outputs are de-energized and the drive is shown as Program Out of Service on the HMI. All alarms are cleared.
Powerup (prescan, first scan)	Processing of modes and alarms on Prescan and Powerup is handled by the embedded P_CmdSrc and P_AlarmAdd-On Instructions. See their specifications for details. On Powerup, the drive is treated as if it had been Commanded to Stop.
Postscan (SFC Transition)	No SFC Postscan logic is provided.

Programming Example

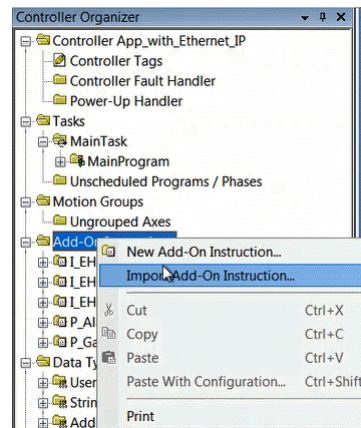
Import the EDS File

It is required to import the Anybus EDS file in order to include the Anybus interface module within the drive in project I/O configuration. See publication [6000-SR007C, PowerFlex 6000 Medium Voltage Variable Frequency Drive Communications Specifications](#), for instructions on loading the EDS file.

Import Device Add-On Instruction

This procedure imports the definitions for the device Add-On Instruction. It is only necessary to import the definitions once per controller.

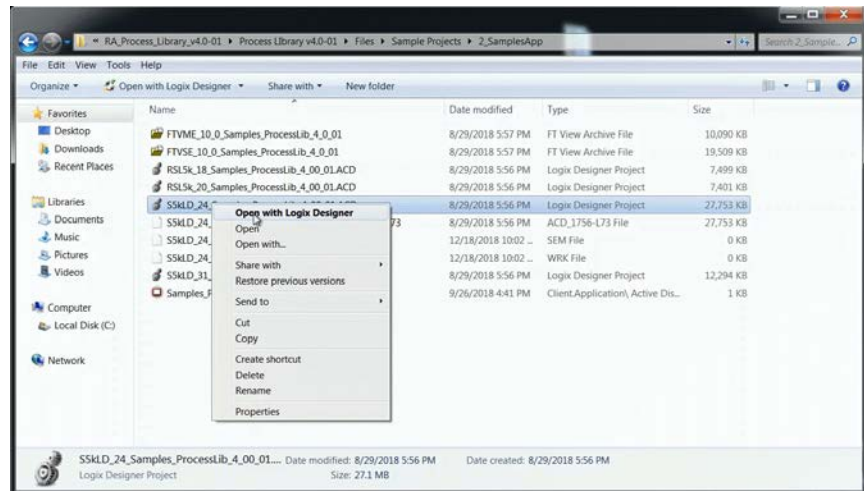
1. In the target Controller Organizer, right-click on Add-On Instructions and choose Import Add-On Instruction.



2. Navigate to the folder that contains the device Add-On Instructions, select the appropriate instruction, and click open.
3. Click OK in the Import Configuration window.
4. The Add-On Instruction is then added to the Controller Organizer.

Copy Fault Code Tags

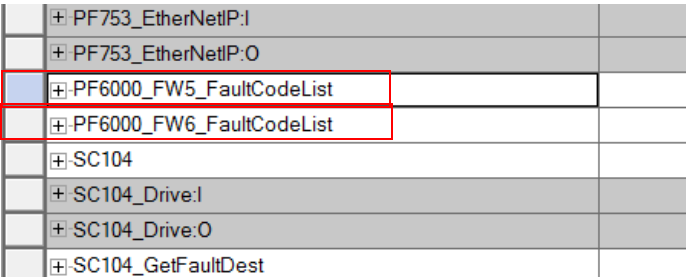
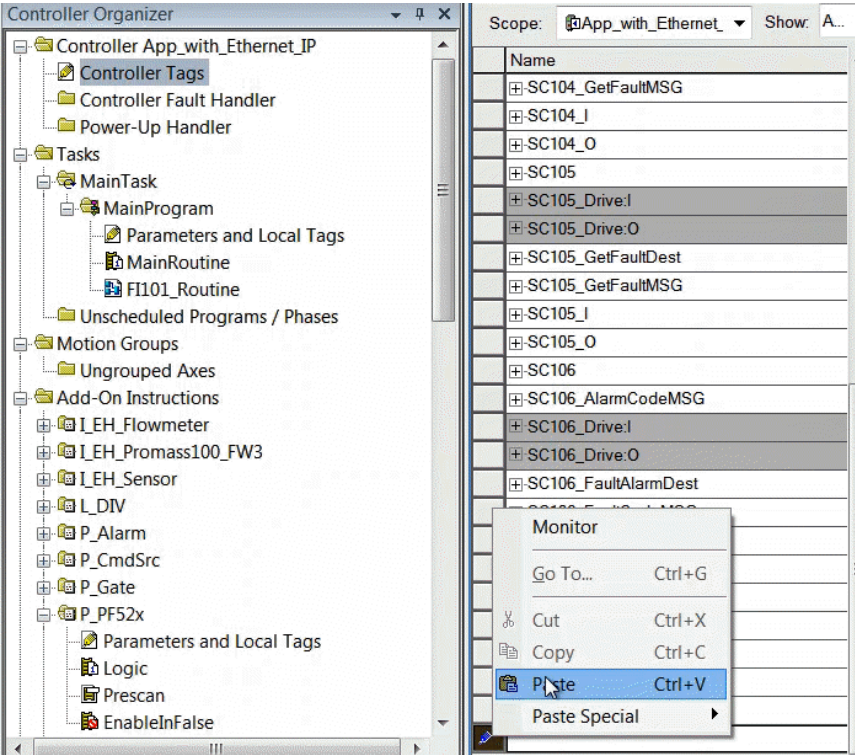
1. Open Project in the Files>Sample Projects>2_SamplesApp Project folder.



2. In the Sample Application, Click Controller Tags in the Controller Organizer. Choose the correct fault code list for PF_6000 based on the drive's firmware revision. There are two options:
 - PF6000_FW5_FaultCodeList (for firmware 5.xxx)
 - PF6000_FW6_FaultCodeList (for firmware 6.xxx and later)

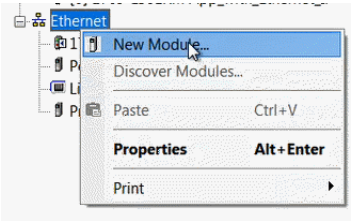
Right Click on the fault code list and choose Copy.

3. In the main project, Click Controller Tags in the Controller Organizer. Right Click at the bottom of the tag list and choose Paste.

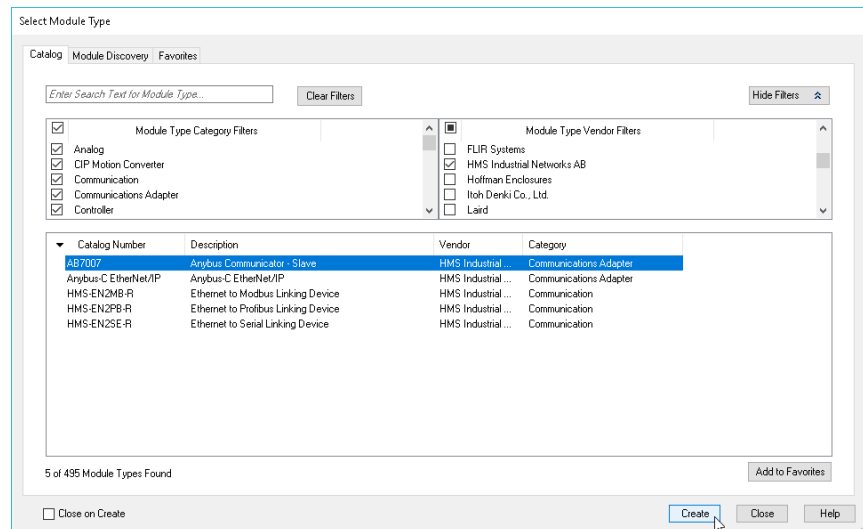


Add the Device to the I/O Tree and Configure the Device

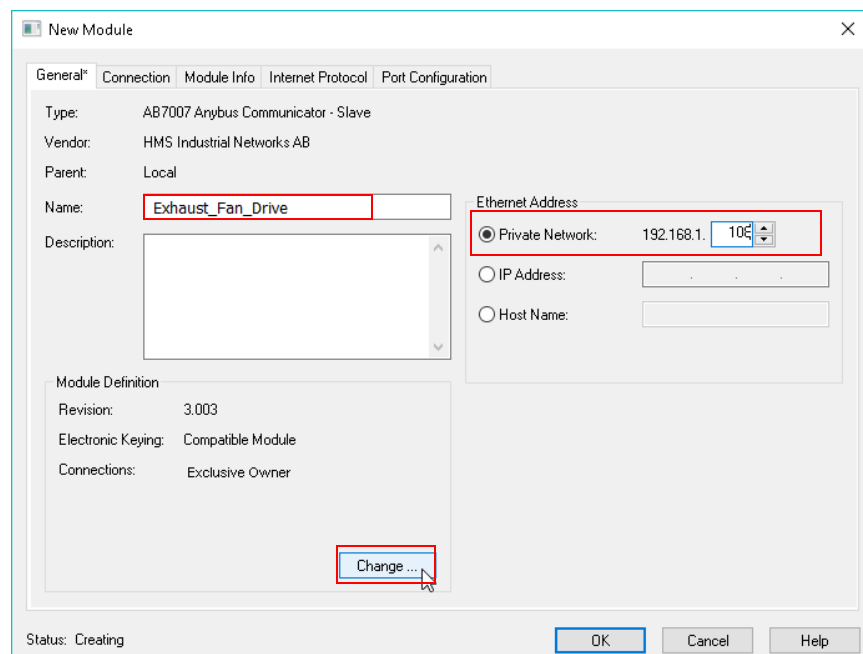
1. In your target application, right-click the Ethernet network in the Controller Organizer and choose New Module.



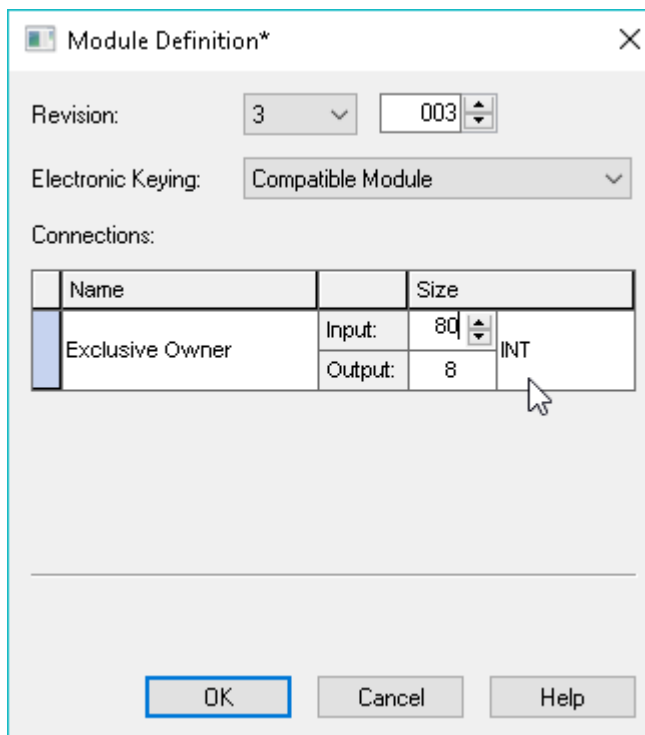
- In the module selection screen, clear all the vendor checkboxes, then check "HMS Industrial Networks AB." Select the AB7007 from the list of HMS modules and click Create.



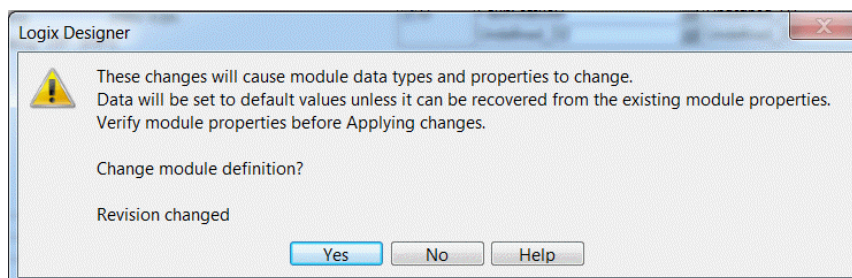
- Change the device name and IP address to match the specifications of your project. Click Change in the Module Definition section.



4. Select "INT" (16-bit signed integer) as the data type. Set the input size to 80 (eighty) INTs, and the output size to 8 (eight) INTs. Click "OK" and "OK."



5. Click YES in the confirmation dialog box to accept the changes to the connection properties.



IMPORTANT The "Data[]" arrays are now 16-bit (signed) integers (INT). The offsets should now match the documentation. For example, the watchdog value going to the drive is now "Exhaust_Fan_Drive:O.Data[4]" - any logic in the controller that talks to the input and output assembly tags will need to have the offsets (array indexes) checked. Use the default drive scaling for speed reference and feedback. This is one-after-the-decimal, for a speed reference of 60.0 Hz, send a value of 600 (which fits in an INT, but not in a SINT, which is a SHORT signed integer, a single byte from -128 to 127).

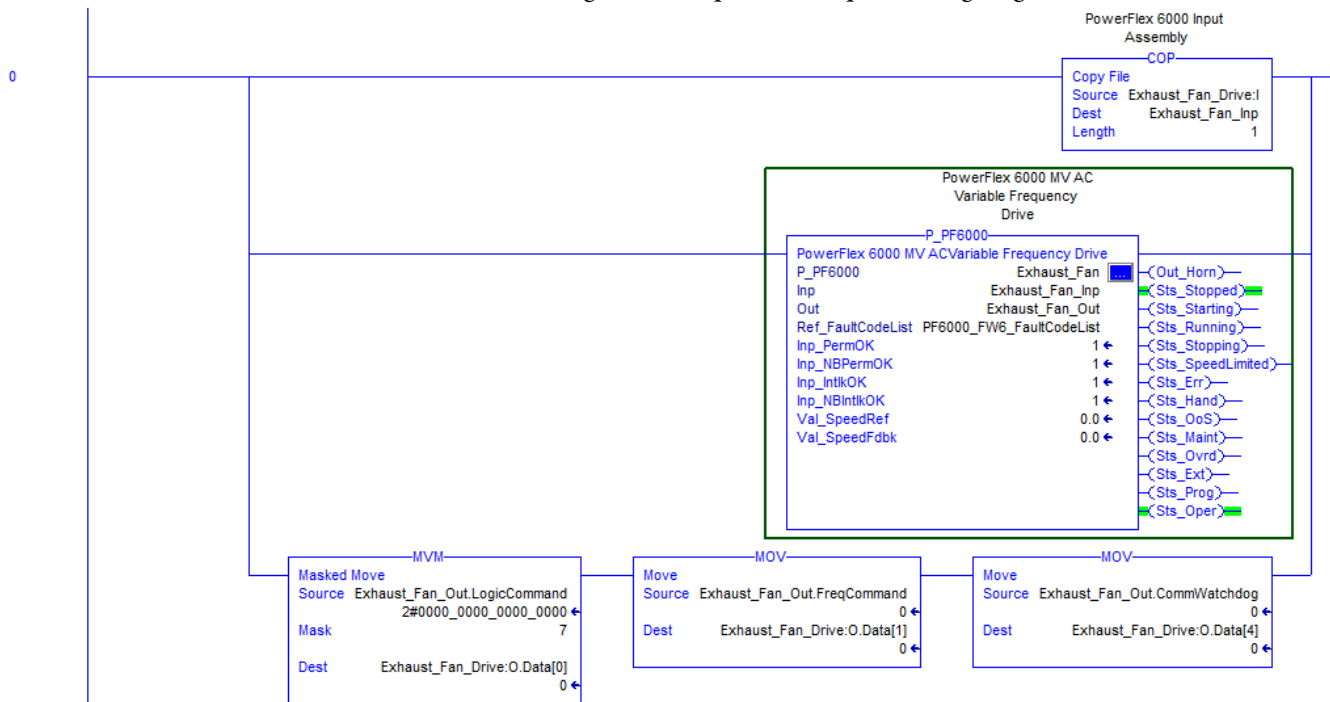
Add the Logic Rung

For the P_PF6000 Add-On Instruction, the rung needs branches which:

- Copy the Input assembly tag data coming FROM the drive into a structured (User-Defined Type) input buffer tag
- Execute the add-on instruction;
- Move the output buffer tag (another User-Defined Type) data to the Output assembly tag data going TO the drive.

The add-on instruction needs to have the correct lookup table for the drive.

The following is an example of a completed rung diagram.



PowerFlex 7000 Drive (P_PF7000)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

-
- IMPORTANT** If you use a drive other than the PowerFlex 7000 drive, use these Add-On Instructions instead:
- P_PF755 for the PowerFlex 755 Drive, or for the PowerFlex 753 with the 20-750-ENETR EtherNet/IP interface
 - P_PF753 for the PowerFlex 753 Drive with the 20-COMM-E EtherNet/IP interface
 - P_PF52x for the PowerFlex 523 or PowerFlex 525 Drive on an EtherNet/IP network
 - P_VSD for third-party drives, drives on other networks, or via hardwired I/O
-

Functional Description

The P_PF7000 instruction provides the following capabilities:

- Control of the drive through the standard P_CmdSrc Add-On Instruction.
- Ability to start and stop the drive and motor, to control the drive speed (via speed reference), to verify whether the drive is running or stopped. You can also monitor the drive run status and speed feedback. Provides alarms and drive shutdown for Fail to Start and Fail to Stop if the feedback does not follow the commanded state within a configured amount of time.
- Reading from the drive, the instruction displays drive faults, drive alarms, conditions that inhibit starting the drive, drive predictive maintenance data, general drive status data, and a number of operating parameters.
- Ability to read a fault code from the drive and provide descriptive text of fault codes.
- Indication of Accelerating, Decelerating, At Speed, Warning, or Alarm status as received from the drive.
- Optional capability to support reversing drives, with commands for forward and reverse rotation, and display of actual rotation direction.
- Input and alarm for a drive fault condition and an output to send a drive fault reset to the drive. Provide a configurable time to pulse the drive fault reset output when a reset command is received.

- Permissives (bypassable and non-bypassable) that are conditions that enable a drive start and Interlocks (bypassable and non-bypassable) that are conditions that stop the drive and help prevent starting. Provide an alarm when an Interlock stops the drive. Provide maintenance personnel with the capability to bypass the bypassable Permissives and Interlocks.
- Capability for maintenance personnel to take the drive out of service.

IMPORTANT This capability is not a substitute for hard lockout/tagout (LOTO) procedures.

- Monitor an I/O fault input and alarm on an I/O fault. The I/O fault condition can optionally de-energize the outputs to the drive, requiring a reset.
- In Override command source, provide an override state input that determines if the override is to run or stop the drive (default = stop), and, if the drive is to run, an override speed reference and direction.
- The instruction provides simulation capability. Outputs to the drive are kept de-energized, but the object can be manipulated as if a working drive were present, including a basic ramp-up of speed feedback value on starting and ramp-down on stopping. The simulated ramp-up-to-speed time is configurable. This capability is often used for activities such as system testing and operator training.

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_PF7000_4.10.00_RUNG.L5X rung import must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Required Drive Configuration

Be certain to configure the drive Datalinks as follows:

- Input Assembly
 - Drive Status (standard)
 - Feedback (standard)
 - Datalinks:

1. Torque Feedback (unfiltered) (%) (Par 489)
 2. Stator Current (% FLA) (Par 340)
 3. Motor Speed (RPM) (Par 363)
 4. Motor Voltage (filtered) (Volts) (Par 362)
 5. Motor Air-Gap Power (%) (Par 346)
 6. User choice #1
 7. User choice #2
 8. User choice #3
- Output Assembly
 - Drive Logic Command (standard)
 - Speed Reference (standard)
 - Datalinks:
 - Drive Logic Command (standard)
 - Speed Reference (standard)
 - All output datalinks are user choice.

InOut Structure

InOut parameters are used to link the Add-On Instruction to external tags that contain necessary data for the instruction to operate. These external tags must be of the data type shown.

Tag Name	Data Type	Description
Inp	P_PF7000_Inp	Common part of PowerFlex 7000 input assembly.
Out	P_PF7000_Out	Common part of PowerFlex 7000 output assembly.
Ref_GetParSrc	INT[72]	List of parameters to get from drive
Ref_GetParMSG	MESSAGE	Message to Get Next Drive Parameter value
Ref_GetParDest	INT[72]	Parameter values retrieved from drive
Ref_DriveFault MSG	MESSAGE	Message to get last fault record.
Ref_DriveAlarm MSG	MESSAGE	Message to get last alarm record.
Ref_FaultAlarm Dest	P_PFComm_FltAlm Recc	Buffer for data from fault record or alarm record message.
Ref_RunTimeMSG	MESSAGE	Message to get elapsed runtime.
Ref_RunTimeDest	LINT	Buffer for data from get elapsed runtime message.

IMPORTANT The user-defined data types (UDTs) are included in the RUNG import that brings in the P_PF7000 Add-On Instruction.

Setup of the MSG Instructions

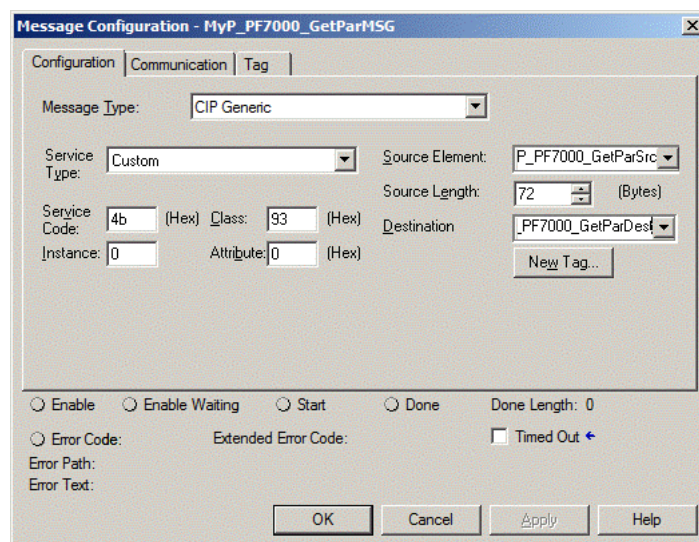
Get PowerFlex 7000 Parameter Values Message (Tags are configured as follows):

- Ref_GetParMSG (control structure for the MSG to get parameter values from the drive)
 - Name the tag “<drive_tag>_GetParMSG”
 - Data Type: MESSAGE
- Ref_GetParSrc (source data for the message: the list of parameters to retrieve)
 - Name the tag “<drive_tag>_GetParSrc”
 - Data Type: INT[36]
- Ref_GetParDest (destination for the parameter data retrieved from the drive)
 - Name this tag “<drive_tag>_GetParDest”
 - Data type: INT[36]

The MSG tag is set up as follows:

- Message Type: CIP Generic
- Service Type: Custom
- Service Code: 16#4b
- Class: 16#93
- Instance: 0
- Attribute: 0
- Source Element: The tag you created for the Ref_GetParSrc tag
- Source Length: 72
- Destination: The tag you created for the Ref_FaultAlarmDest tag
- Path (Communication tab): The drive (select from I/O tree)

Figure 4 - 7000 Parameter Values MSG Configuration



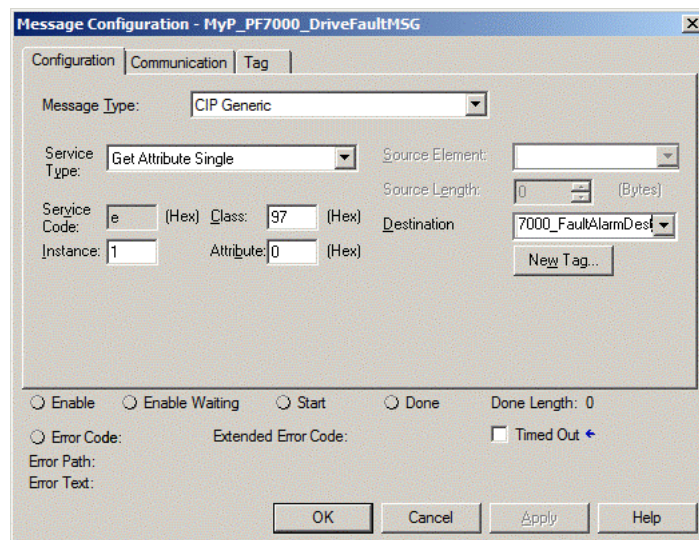
Get PowerFlex 7000 Drive Fault Message (Tags are configured as follows):

- Ref_DriveFaultMSG (control structure for the MSG to get fault data from the drive)
 - Name this tag “<drive_tag>_DriveFaultMSG”
 - Data type: MESSAGE
- Ref_FaultAlarmDest (destination for the fault / alarm data retrieved from the drive)
 - Name this tag “<drive_tag>_FaultAlarmDest”
 - Data type: P_PFComm_FltAlmRec

The MSG tag is set up as follows:

- Message Type: CIP Generic
- Service Type: Get Attribute Single (code 16#0e)
- Class: 16#97
- Instance: 1
- Attribute: 0
- Destination: The tag you created for the Ref_FaultAlarmDest tag
- Path (Communication tab): The drive (select from I/O tree)

Figure 5 - 7000 Drive Fault MSG Configuration



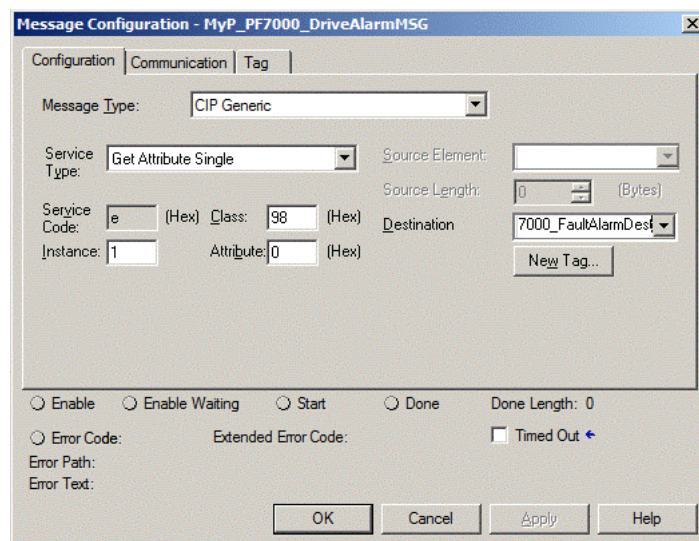
Get PowerFlex 7000 Drive Alarm Message (Tags are configured as follows):

- Ref_DriveAlarmMSG (control structure for the MSG to get fault data from the drive)
 - Name this tag “<drive_tag>_DriveAlarmMSG”
 - Data type: MESSAGE
- Ref_FaultAlarmDest (destination for the fault / alarm data retrieved from the drive, same tag as previous message)
 - Name this tag “<drive_tag>_FaultAlarmDest”
 - Data type: P_PFComm_FltAlmRec

The MSG tag is set up as follows:

- Message Type: CIP Generic
- Service Type: Get Attribute Single (code 16#0e)
- Class: 16#98
- Instance: 1
- Attribute: 0
- Destination: The tag you created for the Ref_FaultAlarmDest tag (same as previous MSG)
- Path (Communication tab): The drive (select from I/O tree)

Figure 6 - 7000 Drive Alarm MSG Configuration



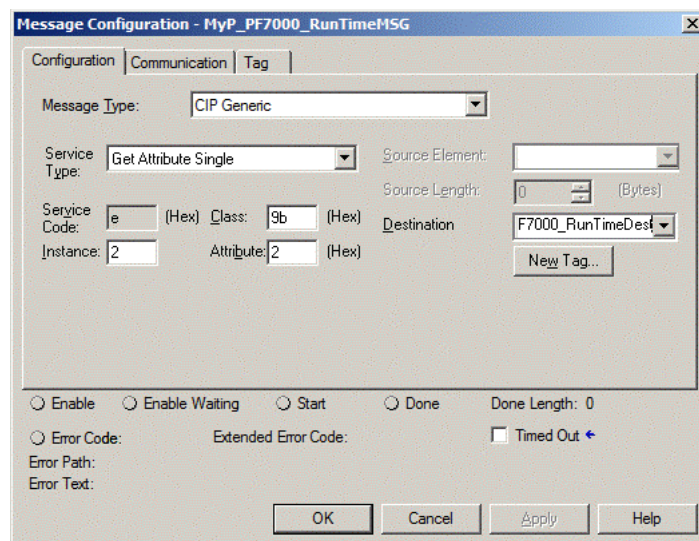
Get PowerFlex 7000 Elapsed Run Time Message (Tags are configured as follows):

- Ref_RunTimeMSG (control structure for the MSG to get elapsed runtime data from the drive)
 - Name this tag “<drive_tag>_DriveFaultMSG”
 - Data type: MESSAGE
- Ref_RunTimeDest (destination for the fault / alarm data retrieved from the drive)
 - Name this tag “<drive_tag>_RunTimeDest”
 - Data type: LINT

The MSG tag is set up as follows:

- Message Type: CIP Generic
- Service Type: Get Attribute Single (code 16#0e)
- Class: 16#9b
- Instance: 2
- Attribute: 2
- Destination: The tag you created for the Ref_RunTimeDest tag
- Path (Communication tab): The drive (select from I/O tree)

Figure 7 - 7000 Elapsed Run Time MSG Configuration



Operations

This section describes the primary operations for this Add-On Instruction.

Alarms


This instruction uses the following alarms, which are implemented by using embedded P_Alarm Add-On Instructions.

Alarm Name	P_Alarm Name	P-Gate Name	Description
Drive Fault	DriveFault	None	Raised when the drive detects a fault and sets its Faulted status bit. Check the Fault Code and description to determine the cause. Issuing a Reset of this object causes a Clear Fault command to be sent to the drive in an attempt to clear the fault.
Fail to Start	FailToStart	None	Raised when the drive has and is using run feedback, an attempt is made to start the drive, and the run feedback does not indicate that the drive is running within the configured time. If Fail to Start is configured as a shed fault, the drive is stopped and a reset is required to start the drive.
Fail to Stop	FailToStop	None	Raised when the drive has and is using run feedback, an attempt is made to stop the drive, and the run feedback does not indicate that the drive stopped within the configured time.
Interlock Trip	IntlkTrip	None	Raised when the drive is running and an interlock 'not OK' condition causes the drive to stop. If interlocks are not bypassed, a bypassable interlock or a non-bypassable interlock 'not OK' condition initiates an interlock trip. If interlocks are bypassed, only a non-bypassable interlock 'not OK' condition initiates an interlock trip.
I/O Fault	IOFault	None	Raised when the Inp_IOFault input is true. This input is used to indicate to the instruction that a communication failure has occurred for its I/O. If the I/O fault is configured as a shed fault, the drive is stopped and not permitted to start until reset.

Parameters of the P_Alarm object can be accessed by using the following convention: [P_Alarm Name].[P_Alarm Parameter].

Simulation

When P_PF7000 is in simulation, the instruction keeps its outputs de-energized and simulates a working drive.

Set the Inp_Sim parameter in the Controller to '1' to enable simulation. When the drive is in simulation, the Simulation or Loopback Test icon  is displayed at the top left of the Operator faceplate.

When in simulation, the object responds to start, stop and jog commands and speed reference changes as if a working drive were being controlled. The configuration parameter Cfg_SimRampT sets the amount of time the simulation takes to ramp from zero speed to maximum and vice versa. The stopped status is asserted when the simulated speed feedback is zero, and running/jogging status is asserted when the simulated speed is greater than zero. There are also configuration bits to determine how speed feedback is calculated based on the speed reference. See the faceplate to see how these configuration bits perform the scaling.

When you have finished in simulation, set the Inp_Sim parameter in the Controller to '0' to return the motor to normal operation.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	Processing for EnableIn False (false rung) is handled the same as if the drive were taken out of service by Command. The drive outputs are de-energized and the drive is shown as Program Out of Service on the HMI. All alarms are cleared.
Powerup (prescan, first scan)	Processing of modes and alarms on Prescan and Powerup is handled by the embedded P_CmdSrc and P_Alarm Add-On Instructions. See the specifications for details. On Powerup, the drive is treated as if it had been Commanded to Stop.
Postscan (SFC Transition)	No SFC Postscan logic is provided.

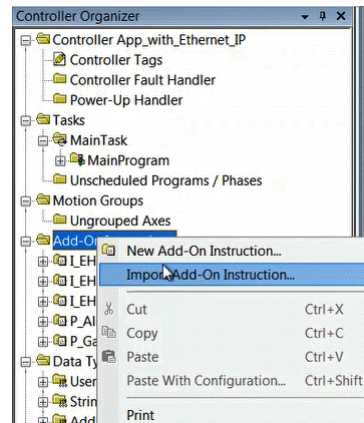
Programming Example

This example will explain how to add the device instruction into your project. Ensure your project is open. For this example we will use P_PF7000.

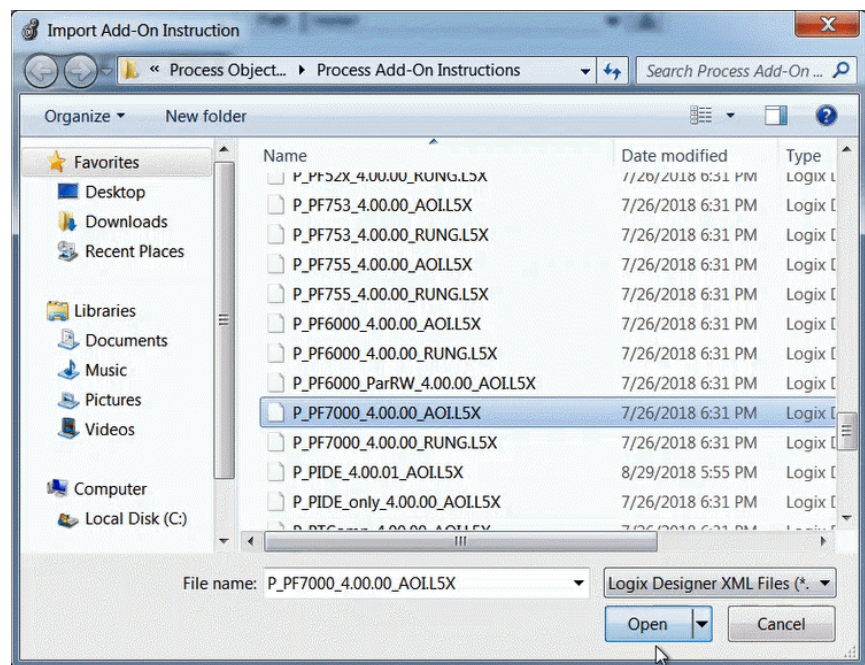
Import Device Add-On Instruction

This procedure imports the definitions for the device Add-On Instruction. It is only necessary to import the definitions once per controller.

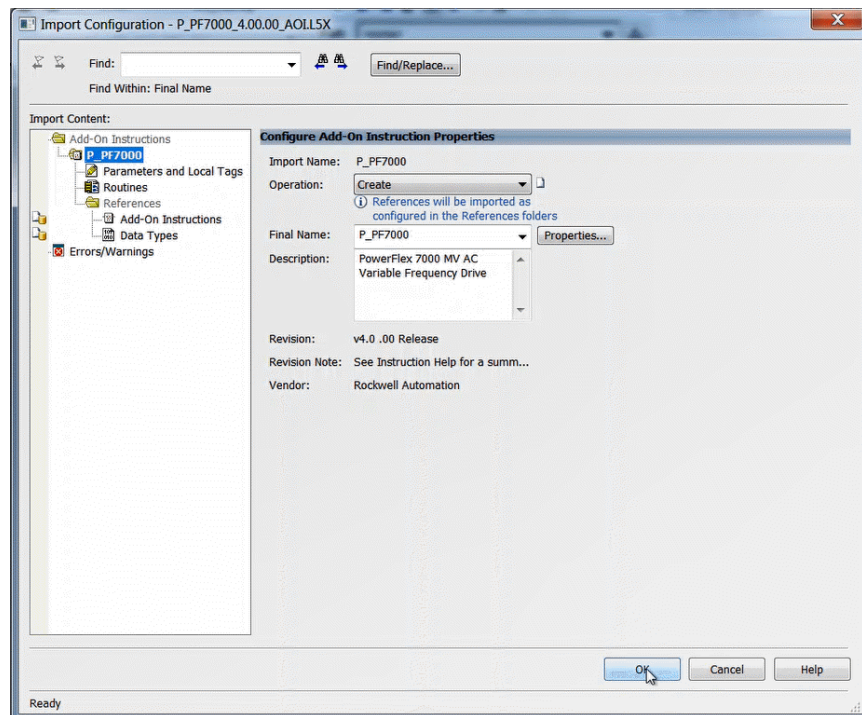
1. In the target Controller Organizer, right-click on Add-On Instructions and choose Import Add-On Instruction.



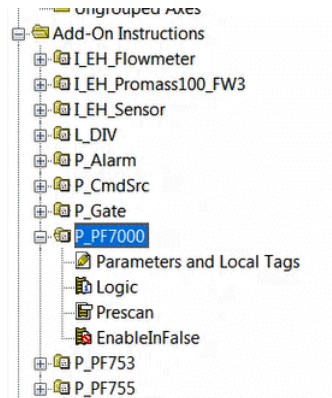
2. Navigate to the folder that contains the device Add-On Instructions, select the appropriate instruction, and click open.



- Click OK in the Import Configuration window.

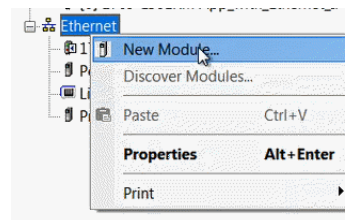


- The Add-On Instruction is then added to the Controller Organizer.

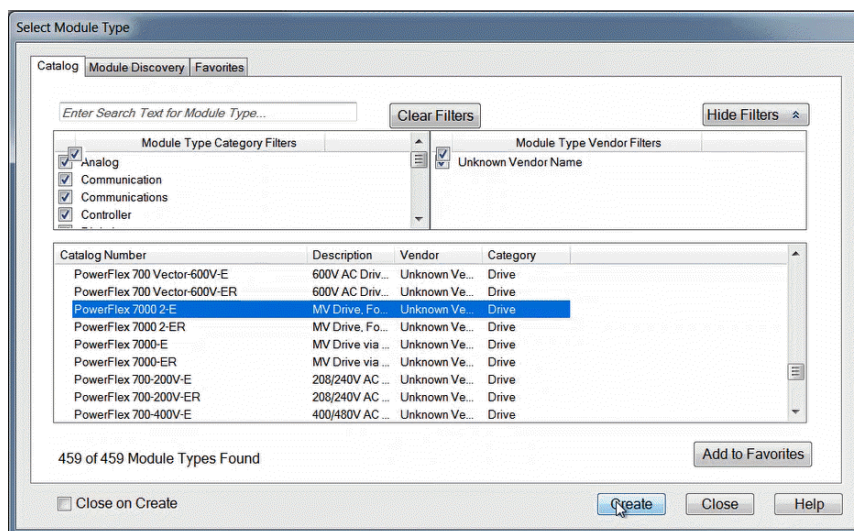


Add the Device to the I/O Tree and Configure the Device

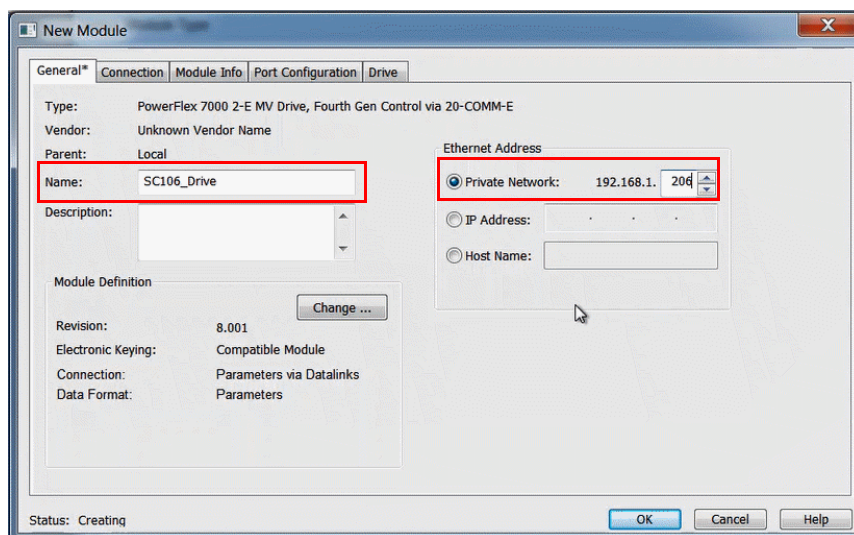
- In your target application, right-click the Ethernet network in the Controller Organizer and choose New Module.



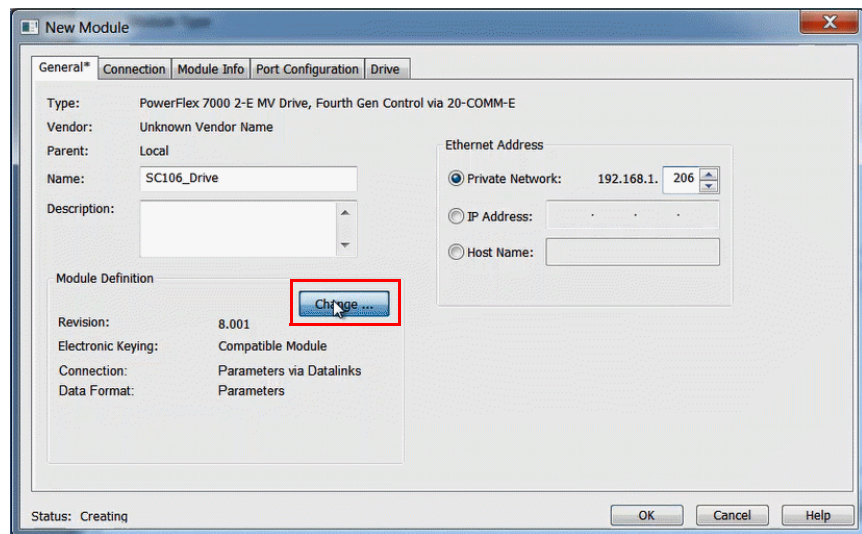
2. Select the Module Type and click Create.



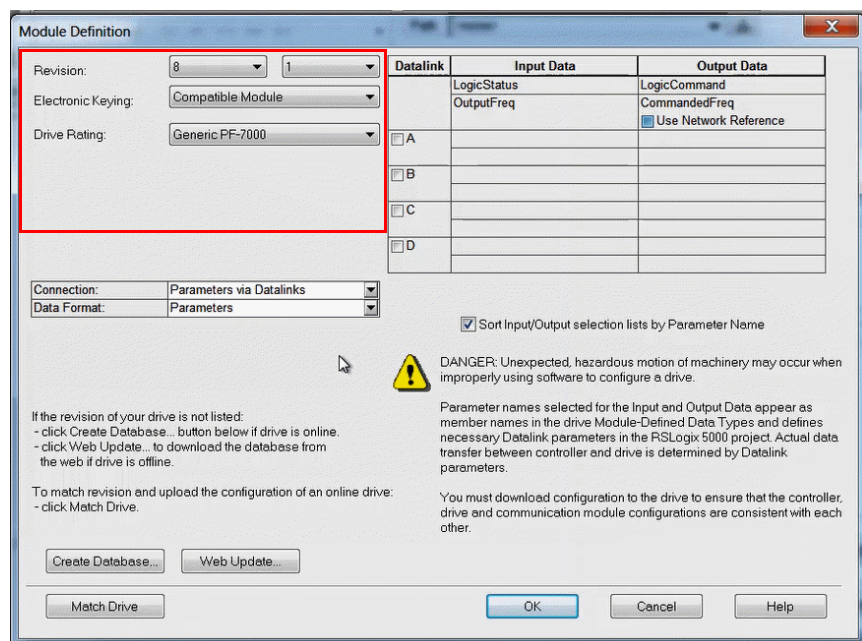
3. Change the device name and IP address to match the specifications of your project.



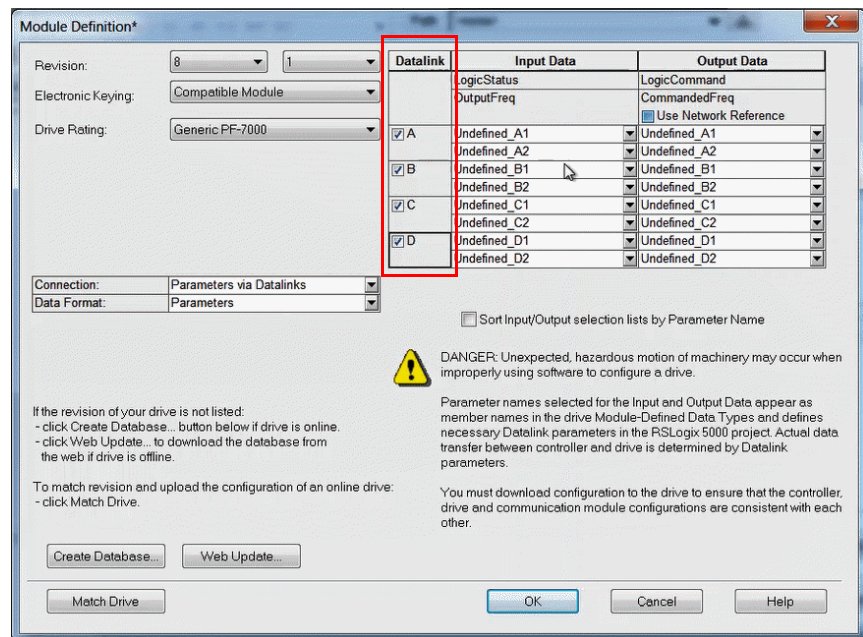
4. Click Change in the Module Definition section.



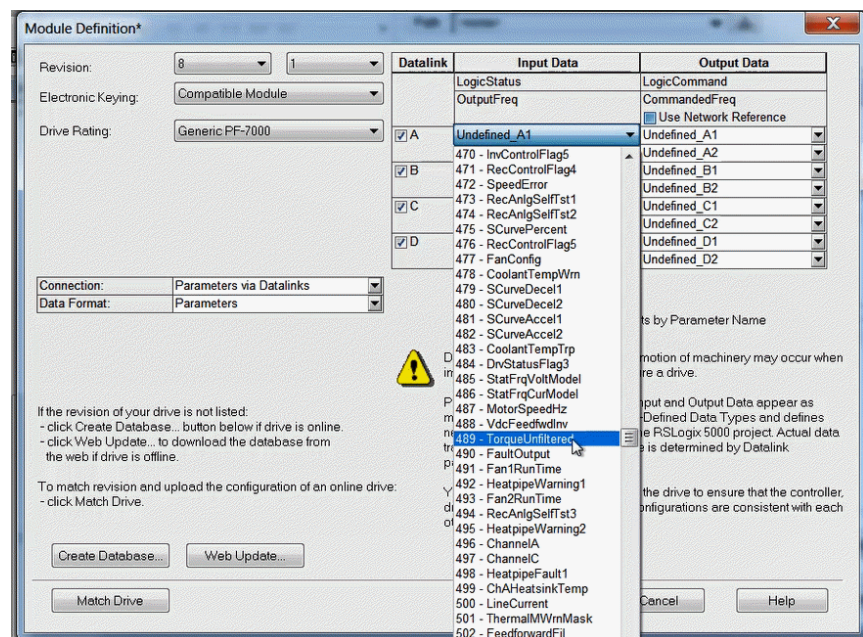
5. Change the information for Revision, Electronic Keying, and Drive Rating to match the specifications of your project.



6. Check the boxes in the Datalink column to add the datalinks.



7. From the pull-down menu, choose the port and parameter for each Datalink and click OK. You will return to the Module Definition dialog box after each Datalink.

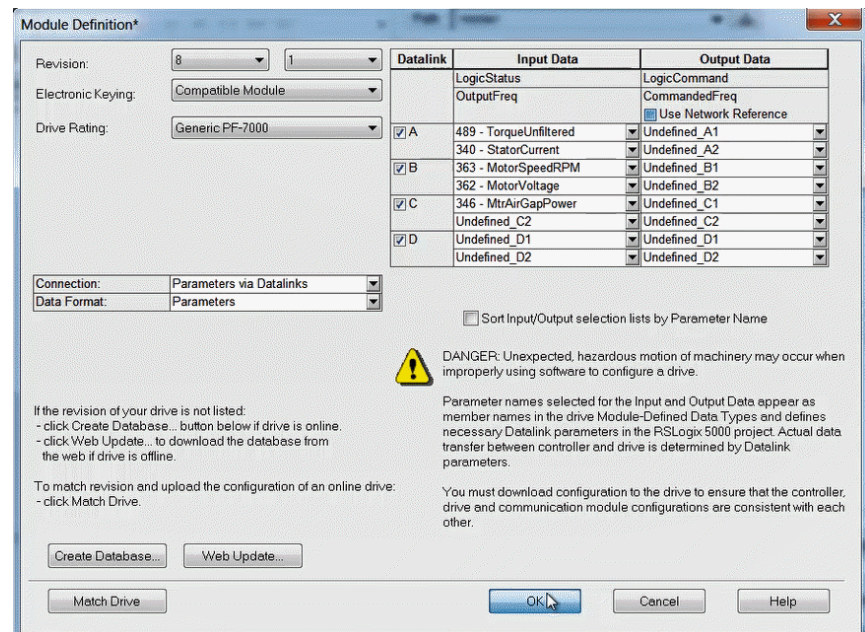


Repeat step 7 for each Datalink.

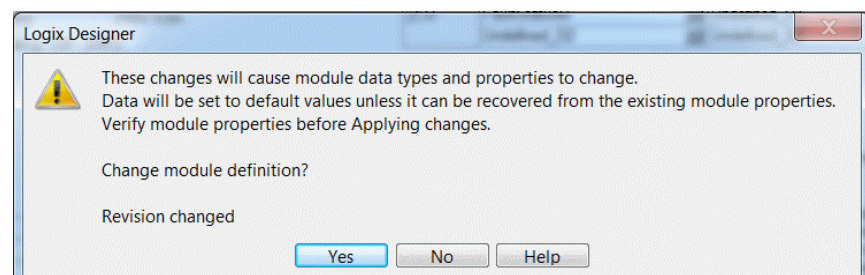
The required DataLinks to add to your project are:

- Input Assembly
 - Drive Status (standard)
 - Feedback (standard)
 - Datalinks:

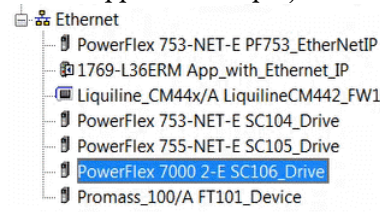
1. Torque Feedback (unfiltered) (%) (Par 489)
 2. Stator Current (% FLA) (Par 340)
 3. Motor Speed (RPM) (Par 363)
 4. Motor Voltage (filtered) (Volts) (Par 362)
 5. Motor Air-Gap Power (%) (Par 346)
 6. User choice #1
 7. User choice #2
 8. User choice #3
- Output Assembly
 - Drive Logic Command (standard)
 - Speed Reference (standard)
 - Datalinks:
 - Drive Logic Command (standard)
 - Speed Reference (standard)
 - All output datalinks are user choice.
8. Once all Datalinks are provisioned click OK in the Module Definition dialog box.



9. Click YES in the confirmation dialog box to accept the changes to the datalinks.




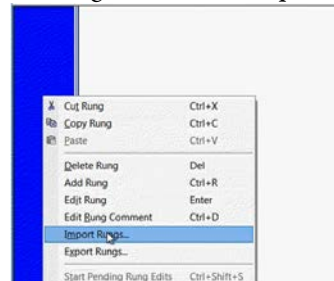
10. The selected device now appears in the project.



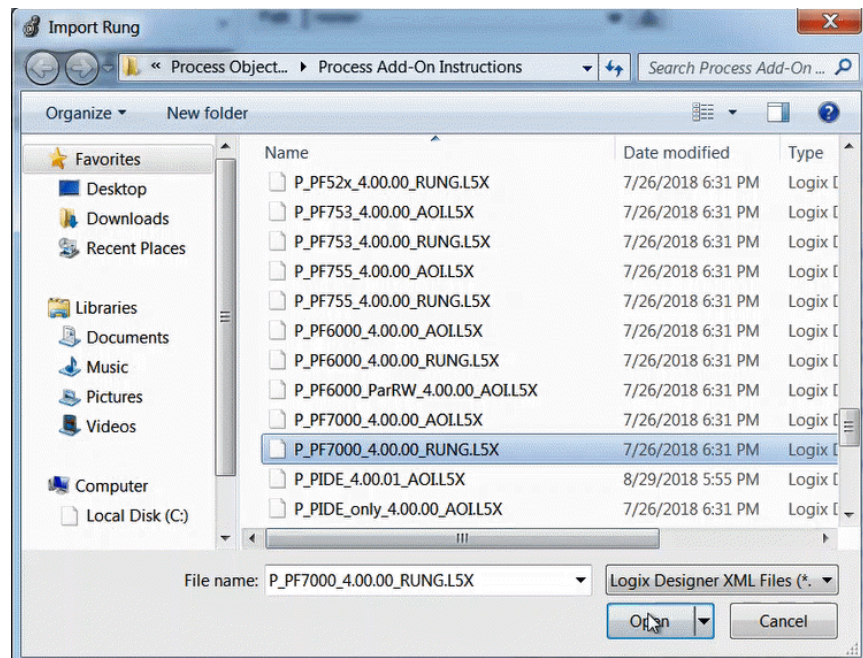
Add the Logic Rung

Follow these steps to import a rung into your project.

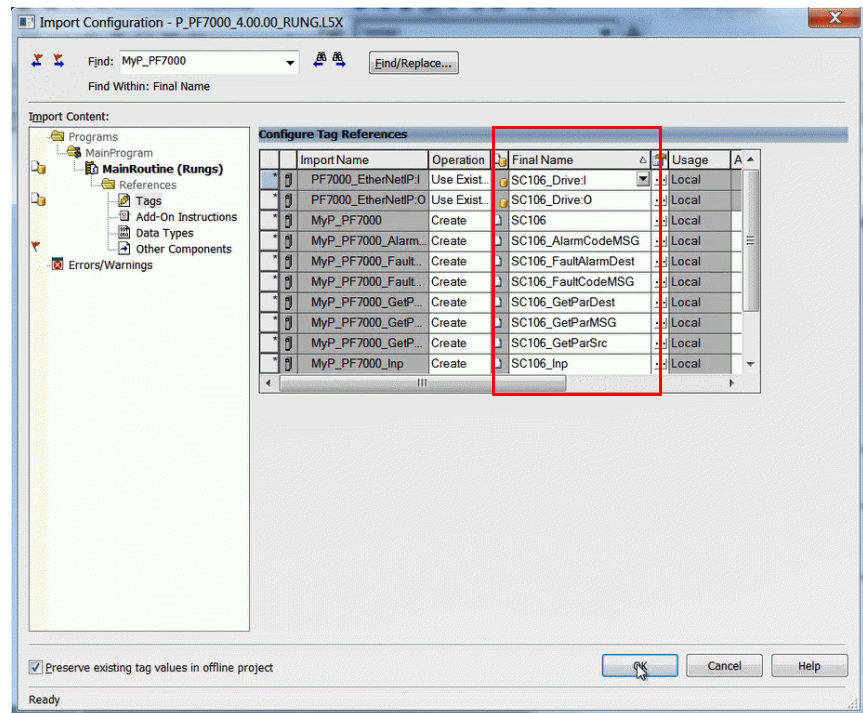
1. On the Controller Organizer, under Tasks, click  in front of Main Task.
2. Double-click Main_Routine to open this ladder logic routine.
3. Right-click one of the rungs and choose Import Rungs.



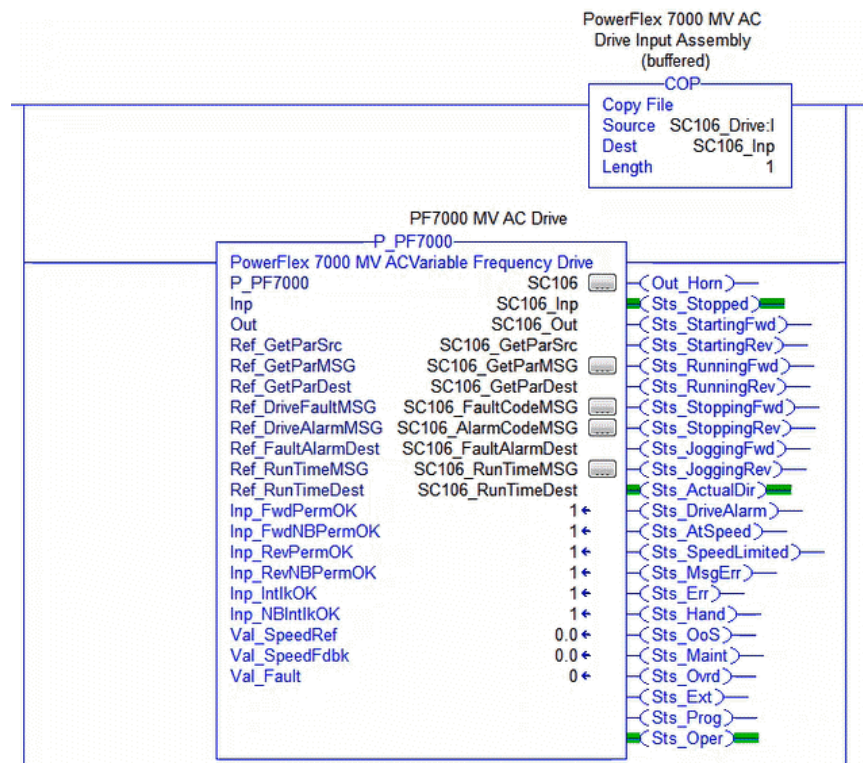
4. On the Import Rungs dialog box, select the device RUNG instruction and click Open.



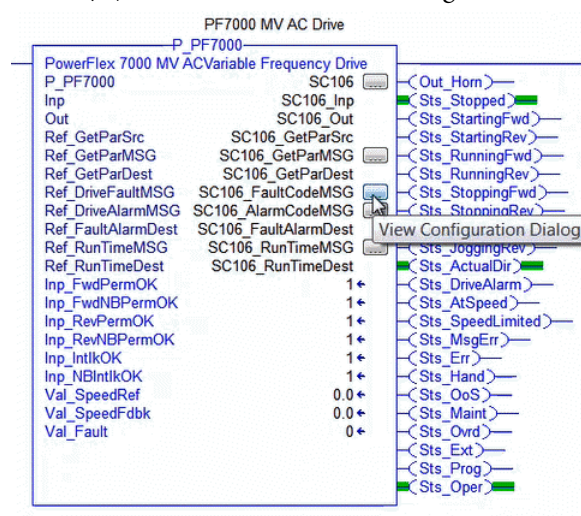
- During the import process, name the tags for the routine in the Import Configuration dialog box. In the Import Content tree, click Tags and type the names of the variables that match your process and the drive name in the Final Name column. Click OK when finished.



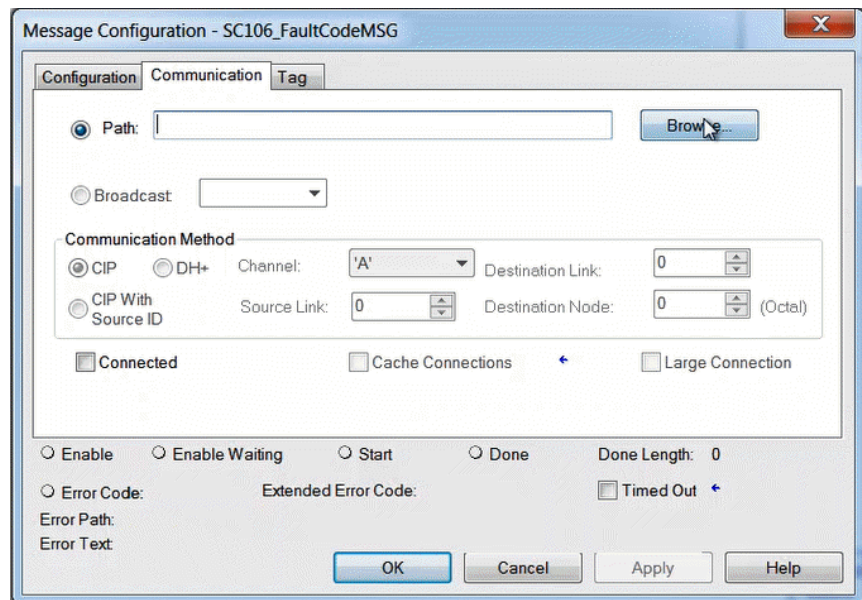
Your ladder logic routine now looks like the example. Observe that the tag names and the drive name are automatically placed in the instruction.



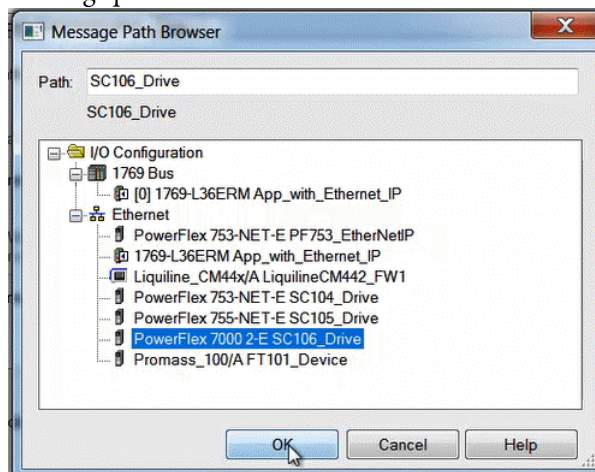
6. Click Browse (...) next to the GetFaultMSG tag to view the configuration.



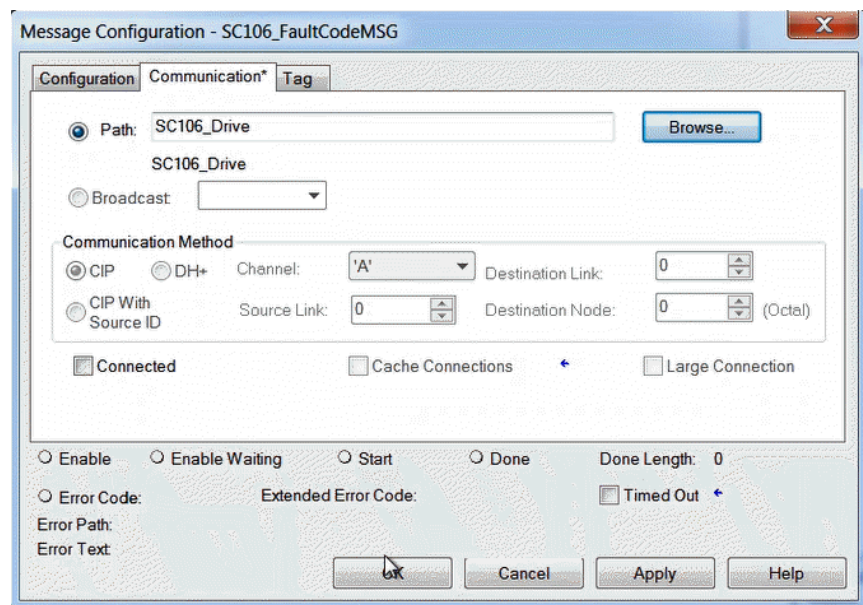
7. Click the Communication tab and then Click Browse.



8. Set the message path to the device and Click OK.



9. Click OK in the Message Configuration dialog box.



10. Repeat steps 6 through 9 for the other MSG tags.

SMC-50 Smart Motor Controller (P_SMC50)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_SMC50 (SMC™-50 Smart Motor Controller) Add-On Instruction controls and monitors a motor via an SMC-50 Smart Starter.

Functional Description

The SMC-50 Smart Motor Controller instruction provides the following capabilities:

- Starting, stopping, and jogging of the motor
- Monitoring of run feedback and display of actual motor status
- Detection of Failure to Start, Failure to Stop, and Motor/Starter Fault
- Monitoring of Permissive conditions to allow starting
- Monitoring of Interlock conditions to stop or prevent starting
- Simulation, providing feedback of a working motor and starter while disabling outputs
- Monitoring of I/O communication faults
- Capability for maintenance personnel to take the drive out of service.

IMPORTANT This capability is not a substitute for hard lockout/tagout (LOTO) procedures.

- Alarms for Fail to Start, Fail to Stop, Interlock Trip, Motor/Starter Fault, and I/O Fault
- Option to reset faults and alarms automatically when an operator commands a motor to start or stop
- Supports HMI 'breadcrumbs' for Alarm Inhibited, Bad Configuration, Not Ready, and Maintenance Bypass Active
- 'Available' status for use by automation logic to know whether motor can be controlled by other objects

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_SMC50_4.10.00_AOIL5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Required SMC-50 Configuration

Be certain to configure the starter Datalinks as follows:

IMPORTANT 'User Choice' Datalinks are not used by this Add-On Instruction and can be left unused or configured for your application.

- Input Assembly:
 - Logic Status (standard)
 - Feedback (Current-Average) (standard)
 - A1. Real Power (Par 10)
 - A2. Power Factor (Par 17)
 - B1. Motor Thermal Usage (Par 18)
 - B2. Time to Trip (Par 19)
 - C1. Time to Reset (Par 20)
 - C2. Fault Code 1 (latest) (Par 138)
 - D1. User choice #1
 - D2. User choice #2
- Output Assembly:
 - Starter Logic Command (standard)
 - Reference (not used) (standard)
 - All output datalinks are user choice.

InOut Structure

InOut parameters are used to link the Add-On Instruction to external tags that contain necessary data for the instruction to operate. These external tags must be of the data type shown.

Tag Name	Data Type	Description
Inp	P_SMC50_Inp	Common part of the SMC-50 Input assembly.
Out	P_SMC50_Out	Common part of the SMC-50 Output assembly.
Ref_FaultCodeList	P_DescList [1]	Array tag containing list of fault codes (DINT) and their descriptions (STRING_40).

IMPORTANT The user-defined data types (UDTs), the I/O assembly tags, and the Array tag containing the list of SMC-50 Smart Motor Controller fault codes and descriptions are included in the rung import. The rung import brings in the P_SMC50 Add-On Instruction.

The following figure shows the drive fault table tags that are in each template.

Scope: ProcessObjects_4		Show: All Tags		PF6000	
Name	Value	Force Mask	Style	Data Type	Description
+ SMC50_FaultCodeList	{...}	{...}		P_DescList[68]	SMC-50 Smart Motor Controller Fault Code List
+ SMCFlex_FaultCodeList	{...}	{...}		P_DescList[48]	SMCFlex Soft Starter Fault Codes and Descriptions
+ SN_MSG_Ctrl	{...}	{...}		MESSAGE	

To display full descriptions for the SMC-50 starter, you must enter the name of the Fault Code List (first column) in the P_SMC50 Ref_FaultCodeList parameter.

Each fault code list has preset codes and descriptions that provide human-readable descriptions of starter fault conditions.

Scope: ProcessObjects_4		Show: All Tags		PF6000	
Name	Value	Force Mask	Style	Data Type	Description
- SMC50_FaultCodeList	{...}	{...}		P_DescList[68]	SMC-50 Smart Motor Controller Fault Code List
- SMC50_FaultCodeList[0]	{...}	{...}		P_DescList	SMC-50 Smart Motor Controller Fault Code List Code / Description List Entry
+ SMC50_FaultCodeList[0].Code	0		Decimal	DINT	Code / Description List Entry Code for which to look up Description
+ SMC50_FaultCodeList[0].Desc	'Check starter manual for this...	{...}		STRING_40	Code / Description List Entry Description for given Code
- SMC50_FaultCodeList[1]	{...}	{...}		P_DescList	SMC-50 Smart Motor Controller Fault Code List Code / Description List Entry
+ SMC50_FaultCodeList[1].Code	1		Decimal	DINT	Code / Description List Entry Code for which to look up Description
+ SMC50_FaultCodeList[1].Desc	'Line Loss Phase A'	{...}		STRING_40	Code / Description List Entry Description for given Code
- SMC50_FaultCodeList[2]	{...}	{...}		P_DescList	SMC-50 Smart Motor Controller Fault Code List Code / Description List Entry
+ SMC50_FaultCodeList[2].Code	2		Decimal	DINT	Code / Description List Entry Code for which to look up Description
+ SMC50_FaultCodeList[2].Desc	'Line Loss Phase B'	{...}		STRING_40	Code / Description List Entry Description for given Code
+ SMC50_FaultCodeList[3]	{...}	{...}		P_DescList	SMC-50 Smart Motor Controller Fault Code List Code / Description List Entry

For a list of fault codes for the SMC-50 Smart Motor Controller, see the SMC-50 Solid-state Smart Motor Controller User Manual, publication [150-UM011](#).

Operations

This section describes the primary operations for this Add-On Instruction.

IMPORTANT See [Appendix A](#) for Command Sources and Simulation types.

Alarms


This instruction uses the following alarms, which are implemented by using embedded P_Alarm Add-On Instructions.

Alarm Name	P_Alarm Name	P-Gate Name	Description
Fail to Start	FailToStart	None	Raised when the SMC has and is using run feedback, an attempt is made to start the drive, and the run feedback does not indicate that the drive is running within the configured time. If Fail to Start is configured as a shed fault, the drive is stopped and a reset is required to start the drive.
Fail to Stop	FailToStop	None	Raised when the SMC has and is using run feedback, an attempt is made to stop the drive, and the run feedback does not indicate that the drive stopped within the configured time.
Interlock Trip	IntlkTrip	None	Raised when the SMC is running and an interlock 'not OK' condition causes the drive to stop. If interlocks are not bypassed, a bypassable interlock or a non-bypassable interlock 'not OK' condition initiates an interlock trip. If interlocks are bypassed, only a non-bypassable interlock 'not OK' condition initiates an interlock trip.
I/O Fault	IOFault	None	Raised when the Inp_IOFault input is true. This input is used to indicate to the instruction that a communication failure has occurred for its I/O. If the I/O fault is configured as a shed fault, the drive is stopped and not permitted to start until reset.
Motor Fault	MotorFault	None	Raised when the Smart Motor Controller detects a fault and sets its Faulted status bit. Check the Fault Code and description to determine the cause. Issuing a Reset of this object causes a Clear Fault command to be sent to the Smart Motor Controller in an attempt to clear the fault.

Parameters of the P_Alarm object can be accessed by using the following convention: [P_Alarm Name].[P_Alarm Parameter].

Simulation

When P_SMC50 is in simulation, the instruction keeps its outputs de-energized and simulates a working motor.

Set the Inp_Sim parameter in the Controller to '1' to enable simulation. When the motor is in simulation, the Simulation or Loopback Test icon  is displayed at the top left of the Operator faceplate.

When in simulation, the object responds to start, stop and jog commands as if a working motor were being controlled. The configuration parameter Cfg_SimFdbkT sets the number of seconds for the simulation to echo back the running or stopped status. For example, if Cfg_SimFdbkT is set to 2.0 seconds and you stop the motor, the simulation will show a “Stopping” status for 2 seconds before showing “Stopped.”

When you have finished in simulation, set the Inp_Sim parameter in the Controller to '0' to return the motor to normal operation.

Execution

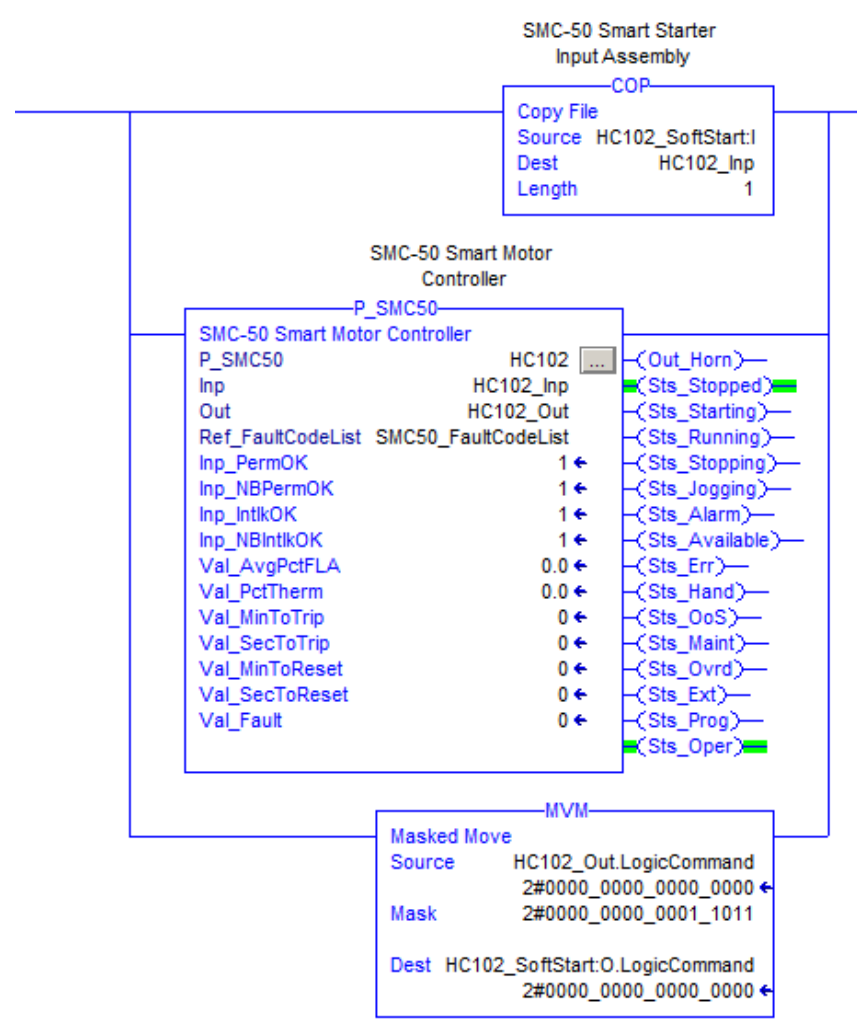
The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	Processing for EnableIn False (false rung) is handled the same as if the motor were taken out of service by Command. The motor outputs are de-energized and the motor is shown as Program Out of Service on the HMI. All alarms are cleared.
Powerup (prescan, first scan)	Any commands received before first scan are discarded. Embedded P_Alarm instructions are handled in accordance with their standard power-up procedures. See the P_Alarm Reference Manual for more information.
Postscan (SFC transition)	No SFC postscan logic is provided.

Programming Example

The following example shows the P_SMC50 in a ladder context. Here, ladder logic is used to copy the Module Defined Data Type for the SMC-50 Smart Starter (AB:SMC_B67CD85C:I:2) to the User Defined Type for the SMC-50 Smart Starter Input Assembly (P_SMC50_Inp).

A complete ladder example is shown below. The Output Value HC102_Out.LogicCommand (User Defined Type P_SMC50_Out) is moved to tag P20_Motor_SoftStart:O.LogicCommand (Module Defined Type AB:SMC_1EB356D5:O:2).



SMC Flex Smart Motor Controller (P_SMCFlex)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

This instruction controls and monitors a motor via an SMC™ Flex Smart Starter.

Functional Description

This SMC Flex Smart Motor Controller provides the following:

- Starting and stopping of the motor
- Monitoring of run feedback and display of actual motor status
- Detection of Failure to Start, Failure to Stop, and Motor/Starter Fault
- Monitoring of Permissive conditions to allow starting
- Monitoring of Interlock conditions to stop/prevent starting
- Capability for maintenance personnel to take the drive out of service.

IMPORTANT This capability is not a substitute for hard lockout/tagout (LOTO) procedures.

- Simulation, which provides feedback of a working motor and starter while disabling outputs
- Monitoring of I/O communication faults
- Alarms for Fail to Start, Fail to Stop, Interlock Trip, Motor/Starter Fault, and I/O Fault
- Option to reset faults and alarms automatically when operator commands motor to start or stop
- Supports HMI 'breadcrumbs' for Alarm Inhibited, Bad Configuration, Not Ready, and Maintenance Bypass Active
- 'Available' status for use by automation logic to know whether the motor can be controlled by other objects

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_SMCFlex_4.10.00_AOIL5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Required SMC Flex Configuration

Make sure that you configure the starter Datalinks as follows:

IMPORTANT 'User Choice' Datalinks are not used by this Add-On Instruction and can be left unused or configured for your application.

- Input Assembly
 - Logic Status (standard)
 - Feedback (Current in Phase A) (standard)
 - A1. Current in Phase B (Par 5)
 - A2. Current in Phase C (Par 6)
 - B1. Watt Meter (Par 7)
 - B2. Power Factor (Par 11)
 - C1. Motor Thermal Usage (Par 12)
 - C2. Fault Code 1 (latest) (Par 124)
 - D1. Motor Full Load Amperes (Par 46)
 - D2. User choice
- Output Assembly
 - Starter Logic Command (standard)
 - Reference (not used) (standard)
 - All output datalinks are user choice.

SMC Flex Smart Motor Controller InOut Structure

InOut parameters are used to link the Add-On Instruction to external tags that contain necessary data for the instruction to operate. These external tags must be of the data type shown.

Tag Name	Data Type	Description
Inp	P_SMCFlex_In P	Common part of the SMC Flex Input assembly.
Out	P_SMCFlex_O ut	Common part of the SMC Flex Output assembly.
Ref_FaultCode List	P_DescList [1]	Array tag containing list of fault codes (DINT) and their descriptions (STRING_40).

IMPORTANT The user-defined data types (UDTs), the I/O assembly tags, and the Array tag containing the list of SMC Flex Smart Motor Controller fault codes and descriptions are included in the Rung import that brings in the P_SMCFlex Add-On Instruction.

The following figure shows the drive fault table tags that are in each template.

Scope: ProcessObjects_4 Show: All Tags PF6000						
Name	Value	Force Mask	Style	Data Type	Description	
+ SMCFlex_FaultCodeList	{...}	{...}		P_DescList[48]	SMCFlex Soft Starter Fault Codes and Descriptions	
+ SN_MSG_Ctrl	{...}	{...}		MESSAGE		
+ SN_MSG_Data	16#0012_f9bb		Hex	DINT		
+ SV3_Sim_ClosedT	{...}	{...}		TIMER		
+ SV3_Sim_ClosingT	{...}	{...}		TIMER		

To display fault descriptions for the SMC Flex Starter, you must enter the name of the Fault Code list (first column) in the P_SMCFlex Ref_FaultCodeList parameter.

Each fault code list has preset codes and descriptions that provide human-readable descriptions of starter fault conditions.

Scope: ProcessObjects_4 Show: All Tags PF6000						
Name	Value	Force Mask	Style	Data Type	Description	
- SMCFlex_FaultCodeList	{...}	{...}		P_DescList[48]	SMCFlex Soft Starter Fault Codes and Descriptions	
- SMCFlex_FaultCodeList[0]	{...}	{...}		P_DescList	SMCFlex Soft Starter Fault Codes and Descriptions Code / Description List Entry	
+ SMCFlex_FaultCodeList[0].Code	0		Decimal	DINT	Code / Description List Entry Code for which to look up Description	
+ SMCFlex_FaultCodeList[0].Desc	*Check starter manual for this...	{...}		STRING_40	Code / Description List Entry Description for given Code	
- SMCFlex_FaultCodeList[1]	{...}	{...}		P_DescList	SMCFlex Soft Starter Fault Codes and Descriptions Code / Description List Entry	
+ SMCFlex_FaultCodeList[1].Code	1		Decimal	DINT	Code / Description List Entry Code for which to look up Description	
+ SMCFlex_FaultCodeList[1].Desc	*Fault: Phase A Power / Line L...	{...}		STRING_40	Code / Description List Entry Description for given Code	
- SMCFlex_FaultCodeList[2]	{...}	{...}		P_DescList	SMCFlex Soft Starter Fault Codes and Descriptions Code / Description List Entry	
+ SMCFlex_FaultCodeList[2].Code	2		Decimal	DINT	Code / Description List Entry Code for which to look up Description	
+ SMCFlex_FaultCodeList[2].Desc	*Fault: Phase B Power / Line L...	{...}		STRING_40	Code / Description List Entry Description for given Code	
+ SMCFlex_FaultCodeList[3]	{...}	{...}		P_DescList	SMCFlex Soft Starter Fault Codes and Descriptions Code / Description List Entry	

For a list of fault codes for the SMC Flex Smart Motor Controller, see the SMC Flex User Manual, publication [150-UM008](#).

Operations

This section describes the primary operations for this Add-On Instruction.

IMPORTANT See [Appendix A](#) for Command Sources and Simulation types.

Alarms


This instruction uses the following alarms, which are implemented by using embedded P_Alarm Add-On Instructions.

Alarm Name	P_Alarm Name	P-Gate Name	Description
Fail to Start	FailToStart	None	Raised when the SMC has and is using run feedback, an attempt is made to start the motor, and the run feedback does not indicate that the motor is running within the configured time. If Fail to Start is configured as a shed fault, the motor is stopped and a reset is required to start the motor.
Fail to Stop	FailToStop	None	Raised when the SMC has and is using run feedback, an attempt is made to stop the motor, and the run feedback does not indicate that the motor stopped within the configured time.
Interlock Trip	IntlkTrip	None	Raised when the SMC is running and an interlock 'not OK' condition causes the motor to stop. If interlocks are not bypassed, a bypassable interlock or a non-bypassable interlock 'not OK' condition initiates an interlock trip. If interlocks are bypassed, only a non-bypassable interlock 'not OK' condition initiates an interlock trip.
I/O Fault	IOFault	None	Raised when the Inp_IOFault input is true. This input is used to indicate to the instruction that a communication failure has occurred for its I/O. If the I/O Fault is configured as a shed fault, the motor is stopped and not permitted to start until reset.
Motor Fault	MotorFault	None	Raised when the Smart Motor Controller detects a fault and sets its Faulted status bit. Check the Fault Code and description to determine the cause. Issuing a Reset of this object causes a Clear Fault command to be sent to the Smart Motor Controller in an attempt to clear the fault.

Parameters of the P_Alarm object can be accessed by using the following convention: [P_Alarm Name].[P_Alarm Parameter].

Simulation

When P_SMCFlex is in simulation, the instruction keeps its outputs de-energized and simulates a working motor.

Set the Inp_Sim parameter in the Controller to '1' to enable simulation. When the motor is in simulation, the Simulation or Loopback Test icon  is displayed at the top left of the Operator faceplate.

When in simulation, the object responds to start, stop and jog commands as if a working motor were being controlled. The configuration parameter Cfg_SimFdbkT sets the number of seconds for the simulation to echo back the running or stopped status. For example, if Cfg_SimFdbkT is set to 2.0 seconds and you stop the motor, the simulation will show a “Stopping” status for 2 seconds before showing “Stopped.”

When you have finished in simulation, set the Inp_Sim parameter in the Controller to '0' to return the motor to normal operation.

Execution

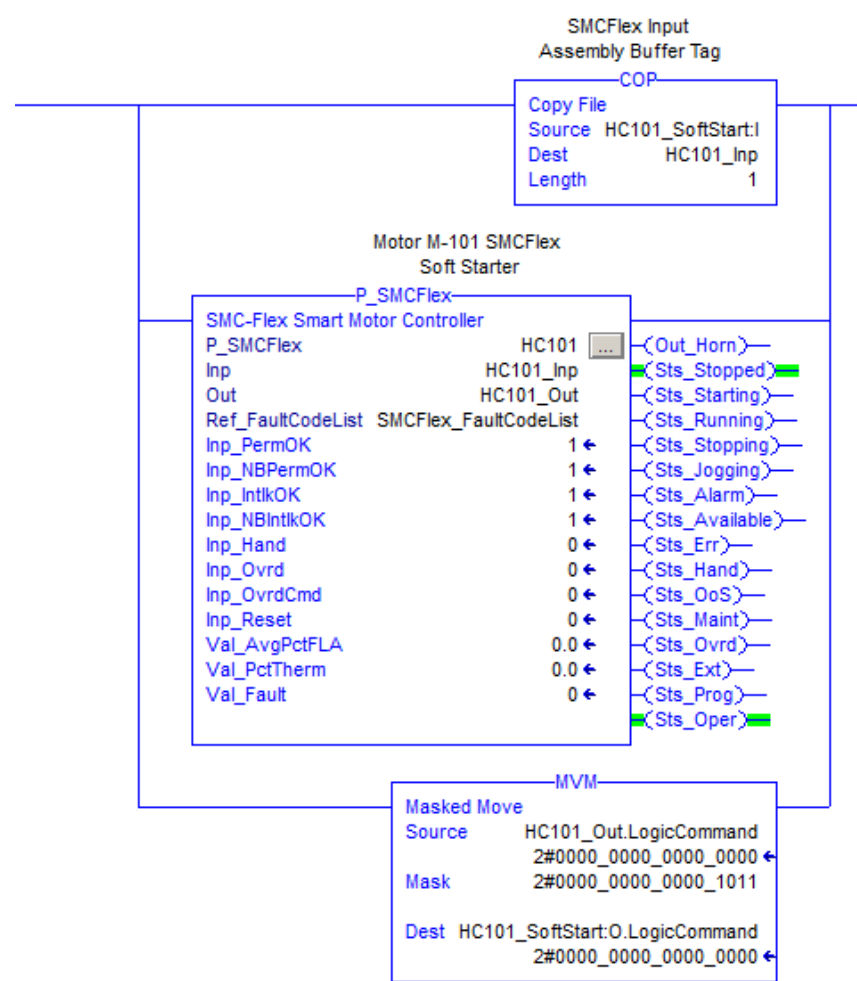
The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	Processing for EnableIn False (false rung) is handled the same as if the motor were taken out of service by Command. The motor outputs are de-energized and the motor is shown as Program Out of Service on the HMI. All alarms are cleared.
Powerup (prescan, first scan)	Any commands received before first scan are discarded. Embedded P_Alarm instructions are handled in accordance with their standard power-up procedures. See the Reference Manual for the P_Alarm Instruction for more information.
Postscan (SFC transition)	No SFC postscan logic is provided.

Programming Example

The following example shows the P_SMCFlex in a ladder context. Here, ladder logic is used to copy the Module Defined Data Type for the SMCFlex Smart Starter (AB:150SMCFlex_BBC3E0FE:I:0) to the User Defined Type for the SMCFlex Smart Starter Input Assembly (P_SMCFlex_Inp).

A complete ladder example is shown below. The Output Value HC101_Out.LogicCommand (User Defined Type P_SMCFlex_Out) is moved to tag P21_Motor_SoftStart:O.LogicCommand (Module Defined Type AB:150SMCFlex_AC39A0CD:O:0).



Variable-speed Drive (P_VSD)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_VSD (Variable Speed Drive) Add-On Instruction is used to operate one variable speed motor by using a drive (AC variable frequency or DC) in a variety of modes, monitoring for fault conditions.

Functional Description

The P_VSD instruction provides the following capabilities:

- Control of the drive through the standard P_CmdSrc Add-On Instruction.
- Ability to start and stop the drive and motor, control the drive speed (via speed reference), and monitor the drive run status and speed feedback to verify that the drive is running or stopped.
- Alarms and drive shut down for Fail to Start and Fail to Stop if the feedback does not follow the commanded state within a configured amount of time. This shut down happens whether the drive has run feedback can be configured at the engineer level or whether to use the run feedback can be configured at the maintenance level.
- Scaling of the speed reference from user (engineering) units, such as RPM, to drive units, such as $32,767 = \text{maximum frequency}$.
- Scaling of the speed feedback from drive units to user (engineering) units and display with suitable units of measure text.
- Optional setting (by Program or Operator) of an output datalink and scaling of this setting from engineering units (such as ramp time in seconds) to drive raw units.
- Optional reading of an input datalink and scaling of this value from drive raw units to engineering units (such as amperes) for display on the HMI.
- Reading from the drive and displaying a fault code, and displaying a text fault description based on the fault code.
- Indication of Accelerating, Decelerating, At Speed, and Warning or Alarm status as received from the drive.
- Optional capability to support reversing drives, with commands for forward and reverse rotation and display of actual rotation direction.
- Input and alarm for drive fault condition and an output to send a drive fault reset to the drive.
- Configurable time to pulse the drive fault reset output when a reset command is received.

- Inputs and outputs formatted to work with any drives commonly used in the Process industries, including, but not limited to, 1336 Plus II, 1395, PowerFlex 4/40/70/700 and PowerFlex DC, in a form that still allows use with non-Allen-Bradley drives via hard-wired I/O.
- Permissives (bypassable and non-bypassable) that are conditions that allow the drive to start, and interlocks (bypassable and non-bypassable) that are conditions that stop the drive and prevent starting.
- Alarm when an interlock stops the drive.
- Capability for maintenance personnel to bypass the bypassable permissives and interlocks.
- Capability for maintenance personnel to take the drive out of service.

IMPORTANT This capability is not a substitute for hard lockout/tagout (LOTO) procedures.

- Monitors I/O communication, and alarms and shuts down on a communication fault.
- Operates with Hand, Maintenance, Override, External, Program, and Operator command source.
- Available status for use by automation logic to determine if other program logic can start and stop the drive.
- Monitor an I/O fault input, alarm on an I/O fault.
- I/O fault condition can optionally de-energize the outputs to the drive, requiring a reset.
- Override State input that determines whether the override is to run or stop the drive (default = stop) and, if the drive is to run, an override speed reference and direction. See the following modes for more information on override.
- Simulation capability, in which the outputs to the drive are kept de-energized, but the object can be manipulated as if a working drive were present, including a basic ramp-up of speed feedback value on starting and ramp-down on stopping. The simulated ramp-up-to-speed time is configurable. This capability is often used for activities such as system testing and operator training.

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_VSD_4.10.00_AOIL5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Required Connections for a Hardwired Drive

[Table 11](#) shows the minimum set of connections to make the P_VSD instruction work with an Allen-Bradley or third-party drive, assuming the drive is non-reversing.

The P_VSD instruction is written around the typical command and status words, speed reference, and speed feedback used with Allen-Bradley PowerFlex drives. Pins are individually available to connect to whatever drive you have, 1336 PLUS II, PowerFlex 4, PowerFlex 40, PowerFlex 70EC, PowerFlex 700, and so on. These can connect over the EtherNet/IP network (in which case the signals have similar, but not identical names), or over DeviceNet, ControlNet, any other network, or over hardwired I/O.

Table 11 - VSD Instruction with a Non-reversing Hardwired Drive

Parameter	Description
Inputs	
Inp_SpeedFdbk	We recommend that you set this value to represent the drive speed; otherwise, the display indicates zero speed. The value can be live speed feedback or it can be a loopback of the speed reference output to the drive. Make sure that you set 'Assume Data Available' if this is a loopback of the output used in Function Block Diagram (FBD) logic.
Inp_Ready	This parameter must be set to 1 for the instruction to allow starting the drive. It's best if Inp_Ready is actively written to '1'. In FBD, connect an IREF that is a '1' (program constant). In LD, code an unconditional OTL to this parameter. This parameter can be 'left at' 1, but if it ever gets zeroed, the Add-On Instruction does not allow a start.
Inp_Running	This parameter must be wired from the run feedback using an IREF with the input bit or an XIC-OTE rung.
Inp_CommandDir, Inp_ActualDir	This parameter must be set to or left at 1 (forward). (See the following for reversing drives.)
Inp_Accelerating, Inp_Decelerating, Inp_Alarm, Inp_Faulted, and Inp_AtSpeed	These parameters are optional. We recommend that you clear or leave at 0 any parameters that are not used, unless you have an input that indicates a drive fault (1=Fault, 0=OK). If you do have an input that indicates a drive fault, we recommend that you wire it to Inp_Faulted.
Outputs	
Out_SpeedRef	Connect to the (analog) output to the drive to set the speed.

Table 11 - VSD Instruction with a Non-reversing Hardwired Drive

Parameter	Description
Out_Run	<p>If the discrete output is 1 to run and 0 to stop, use Out_Run and not Out_Start, Out_Stop, or Out_Jog.</p> <p>In FBD, you can expose the Out_Run pin (use the dialog box opened by the 'ellipsis') and hide the others. Then wire it to an OREF on the output to the drive.</p> <p>In LD, code an XIC of Out_Run and an OTE of the discrete output to the drive.</p> <p>If there are separate outputs for start and stop, wire them to Out_Start and Out_Stop.</p>
Out_ClearFault	If you wired a Drive Fault input and have an output to the drive to attempt to clear the fault, wire it to Out_ClearFault. Otherwise, leave this pin unconnected.
Other outputs	Other outputs can be left unconnected and unreferenced.

[Table 12](#) shows the additional connections to make the P_VSD instruction work with an Allen-Bradley or a third-party drive, assuming the drive is reversing.

Table 12 - Using P_VSD Instruction with a Reversing Hardwired Drive

Parameter	Description
Inp_CommandDir	<p>This parameter must properly reflect the commanded direction. Out_Fwd requests to set the drive to the forward direction, and Out_Rev requests to set the drive to the reverse direction.</p> <p>If the drive itself does not show the commanded direction, logic tied to Inp_CommandDir must be provided to show the result of the requests. This can be done by using a Set-dominant function block or a pair of rungs that set/latch Inp_CommandDir when Out_Fwd is 1 and clear/unlatch Inp_CommandDir when Out_Rev is 1.</p>
Inp_ActualDir	Make sure that this parameter reflects the actual rotation of the motor (1 = Forward, 0 = Reverse).
Inp_SpeedFdbk	<p>This parameter must reflect a non-negative speed (absolute value of speed). A negative value is not interpreted as running reverse.</p> <p>At a minimum, Inp_CommandDir must be handled as stated.</p>

InOut Structure

InOut parameters are used to link the Add-On Instruction to external tags that contain necessary data for the instruction to operate. These external tags must be of the data type shown.

Tag Name	Data Type	Description
Ref_FaultCodeList	P_DescList [1]	Array tag containing list of fault codes (DINT) and their descriptions (STRING_40).

This figure shows the drive fault table tags that are in each template.

Scope: ProcessObjects_4	Show: All Tags	▼	▼	▼	▼
Name	Value	Force Mask	Style	Data Type	Description
PF7xx_FaultCodeList	{...}	{...}		P_DescList[12]	PowerFlex 70 / 700 / 700H Fault Codes and Descriptions
PF40E1	{...}	{...}		AB.PowerFlex...	
PF40E0	{...}	{...}		AB.PowerFlex...	

To have fault codes in P_VSD, you must enter the name of the Fault Code List (first column) in the P_VSD Ref_FaultCodeList parameter.

Each fault code list has preset codes and descriptions. You can use one of the fault codes provided by copying the tag from a template application. You can also create your own by using the provided P_DescList data type.

Scope: ProcessObjects_4	Show: All Tags	▼	▼	▼	▼
Name	Value	Force Mask	Style	Data Type	Description
PF7xx_FaultCodeList	{...}	{...}		P_DescList[12]	PowerFlex 70 / 700 / 700H Fault Codes and Descriptions
PF7xx_FaultCodeList[0]	{...}	{...}		P_DescList	PowerFlex 70 / 700 / 700H Fault Codes and Descriptions Code / Description List Entry
PF7xx_FaultCodeList[0] Code	0		Decimal		Code / Description List Entry Code for which to look up Description
PF7xx_FaultCodeList[0] Desc	'Check drive manual for this f...	{...}		STRING_40	Code / Description List Entry Description for given Code
PF7xx_FaultCodeList[1]	{...}	{...}		P_DescList	PowerFlex 70 / 700 / 700H Fault Codes and Descriptions Code / Description List Entry
PF7xx_FaultCodeList[1] Code	1		Decimal		Code / Description List Entry Code for which to look up Description
PF7xx_FaultCodeList[1] Desc	'Precharge Active'	{...}		STRING_40	Code / Description List Entry Description for given Code
PF7xx_FaultCodeList[2]	{...}	{...}		P_DescList	PowerFlex 70 / 700 / 700H Fault Codes and Descriptions Code / Description List Entry
PF7xx_FaultCodeList[2] Code	2		Decimal		Code / Description List Entry Code for which to look up Description
PF7xx_FaultCodeList[2] Desc	'Auxiliary Input'	{...}		STRING_40	Code / Description List Entry Description for given Code
PF7xx_FaultCodeList[3]	{...}	{...}		P_DescList	PowerFlex 70 / 700 / 700H Fault Codes and Descriptions Code / Description List Entry

For a list of fault codes for a drive, click the Publication number associated with the drive in the following table.

Table 13 - Fault Code Publications

Drive	Publication Number
PowerFlex Digital DC Drive	20P-UM001
PowerFlex 4 Adjustable Frequency AC Drive	22A-UM001
PowerFlex 40 Adjustable Frequency AC Drive	22B-UM001
PowerFlex 400 Adjustable Frequency AC Drive for Fan & Pump Applications	22C-UM001
PowerFlex 40P Adjustable Frequency AC Drive	22D-UM001
PowerFlex 4M Adjustable Frequency AC Drive	22F-UM001
PowerFlex 70 Adjustable Frequency AC Drives	20A-UM001

Table 13 - Fault Code Publications

PowerFlex 700 Adjustable Frequency AC Drive	20B-UM001
PowerFlex 700H Adjustable Frequency AC Drive	20C-PM001
PowerFlex 700S High-Performance AC Drive - Phase II Control	20D-PM001

Operations

This section describes the primary operations for this Add-On Instruction.

IMPORTANT See [Appendix A](#) for Command Sources and Simulation types.

Alarms

This instruction uses the following alarms, which are implemented by using embedded P_Alarm Add-On Instructions.

Alarm Name	P_Alarm Name	P-Gate Name	Description
Drive Fault	DriveFault	None	Raised when the drive detects a fault and sets its Faulted status bit. Check the Fault Code and description to determine the cause. Issuing a Reset of this object causes a Clear Fault command to be sent to the drive in an attempt to clear the fault.
Fail to Start	FailToStart	None	Raised when the drive has and is using run feedback, an attempt is made to start the drive, and the run feedback does not indicate that the drive is running within the configured time. If Fail to Start is configured as a shed fault, the drive is stopped and a reset is required to start the drive.
Fail to Stop	FailToStop	None	Raised when the drive has and is using run feedback, an attempt is made to stop the drive, and the run feedback does not indicate that the drive stopped within the configured time.
Interlock Trip	IntlkTrip	None	Raised when the drive is running and an interlock 'not OK' condition causes the drive to stop. If interlocks are not bypassed, a bypassable interlock or a non-bypassable interlock 'not OK' condition initiates an interlock trip. If interlocks are bypassed, only a non-bypassable interlock 'not OK' condition initiates an interlock trip.
I/O Fault	IOFault	None	Raised when the Inp_IOFault input is true. This input is used to indicate to the instruction that a communication failure has occurred for its I/O. If the I/O Fault is configured as a shed fault, the drive is stopped and not permitted to start until reset.


Parameters of the P_Alarm object can be accessed by using the following convention: [P_Alarm Name].[P_Alarm Parameter].

The Fail to Start and Fail to Stop status and alarms have a configurable delay to allow the run feedback time to align with the commanded output. This delay provides time for the motor to start or stop.

The Fail to Start and I/O fault conditions can be configured to alarm only, or to de-energize the motor (shed). If one of these conditions stops the motor, a reset is required to run.

Simulation

When P_VSD is in simulation, the instruction keeps its outputs de-energized and simulates a working drive.

Set the Inp_Sim parameter in the Controller to '1' to enable simulation. When the drive is in simulation, the Simulation or Loopback Test icon  is displayed at the top left of the Operator faceplate.

When in simulation, the object responds to start, stop and jog commands and speed reference changes as if a working drive were being controlled. The configuration parameter Cfg_SimRampT sets the amount of time the simulation takes to ramp from zero speed to maximum and vice versa. The stopped status is asserted when the simulated speed feedback is zero, and running/jogging status is asserted when the simulated speed is greater than zero. There are also configuration bits to determine how speed feedback is calculated based on the speed reference. See the faceplate to see how these configuration bits perform the scaling.

When you have finished in simulation, set the Inp_Sim parameter in the Controller to '0' to return the motor to normal operation.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	Processing for EnableIn False (false rung) is handled the same as if the drive were taken out of service by Command. The drive outputs are de-energized and the drive is shown as Program Out of Service on the HMI. All alarms are cleared.
Powerup (prescan, first scan)	Processing of command sources and alarms on prescan and powerup is handled by the embedded P_CmdSrc and P_Alarm Add-On Instructions - refer to their specifications for details. On powerup, the drive is treated as if it had been commanded to stop.
Postscan (SFC transition)	No SFC postscan logic is provided.

For more information, see the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#).

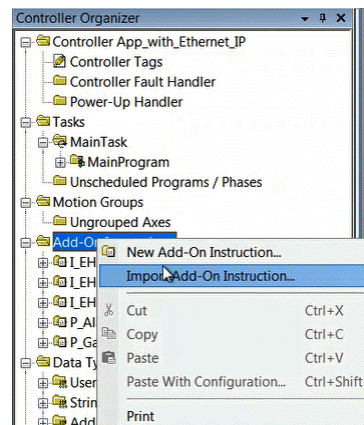
Programming Example

A PowerFlex 700H Adjustable Frequency AC Drive is used in this example.

Import Device Add-On Instruction

This procedure imports the definitions for the device Add-On Instruction. It is only necessary to import the definitions once per controller.

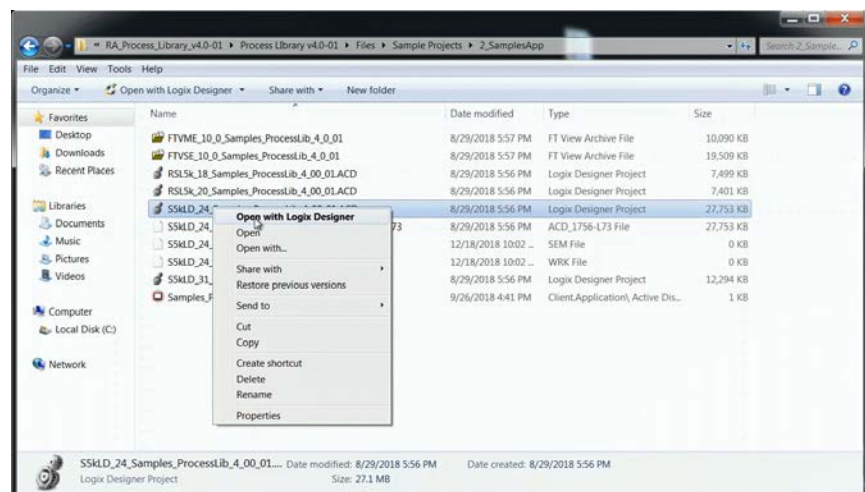
1. In the target Controller Organizer, right-click on Add-On Instructions and choose Import Add-On Instruction.



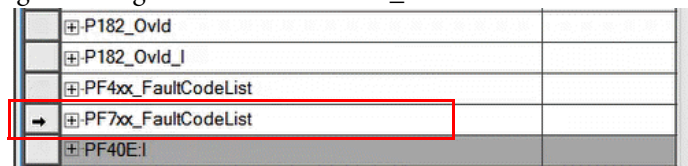
2. Navigate to the folder that contains the device Add-On Instructions, select the appropriate instruction, and click open.
3. Click OK in the Import Configuration window.
4. The Add-On Instruction is then added to the Controller Organizer.

Copy Fault Code Tags

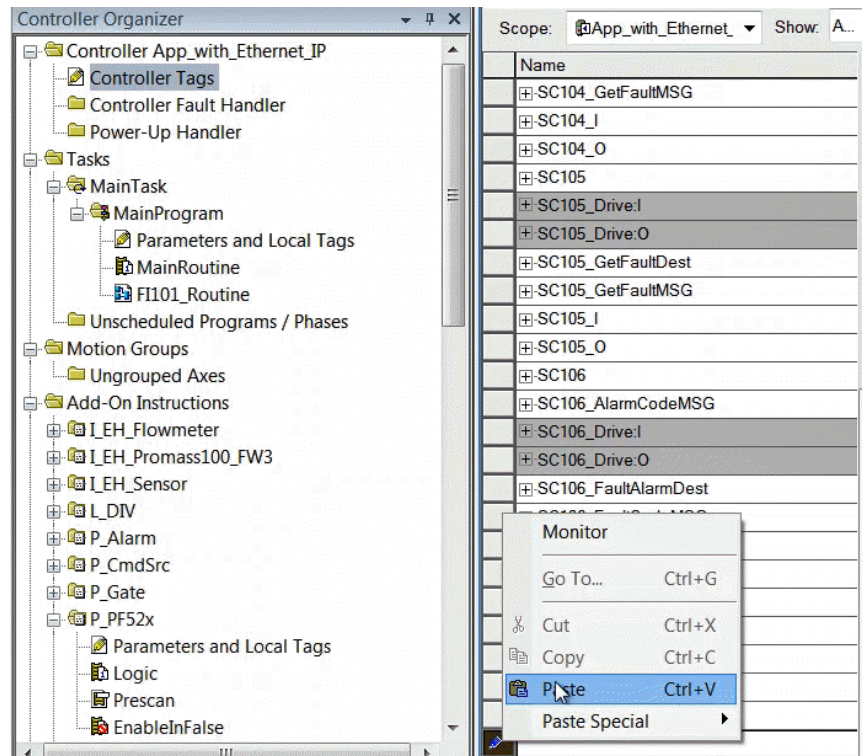
1. Open Project in the Files>Sample Projects>2_SamplesApp Project folder.

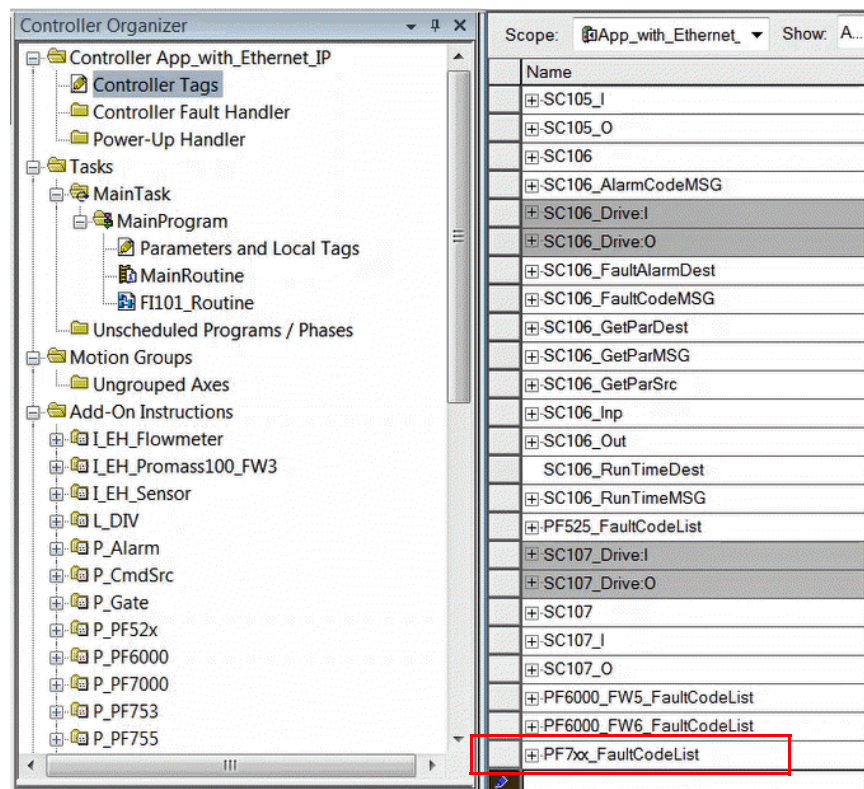


2. In the Sample Application, Click Controller Tags in the Controller Organizer. Right Click on the PF7xx_FaultCodeList and choose Copy.



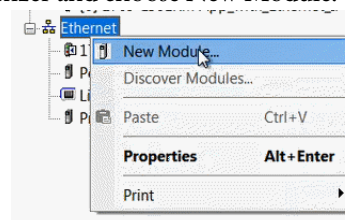
3. In the main project, Click Controller Tags in the Controller Organizer. Right Click at the bottom of the tag list and choose Paste.





Add the Device to the I/O Tree and Configure the Device

1. In your target application, right-click the Ethernet network in the Controller Organizer and choose New Module.



2. Select the Module Type and click Create.
3. Change the device name and IP address to match the specifications of your project.
4. Click Change in the Module Definition section.
5. Change the information for Revision, Electronic Keying, Drive Rating, Rating Options, and Special types to match the specifications of your project.
6. Check the boxes in the Datalink column to add the datalinks.
7. In the Input Data column, click Browse (...). The Parameter Properties dialog box appears.

8. From the pull-down menu, choose the port and parameter for each Datalink and click OK. You will return to the Module Definition dialog box after each Datalink.

Repeat steps 7 and 8 for each Datalink.

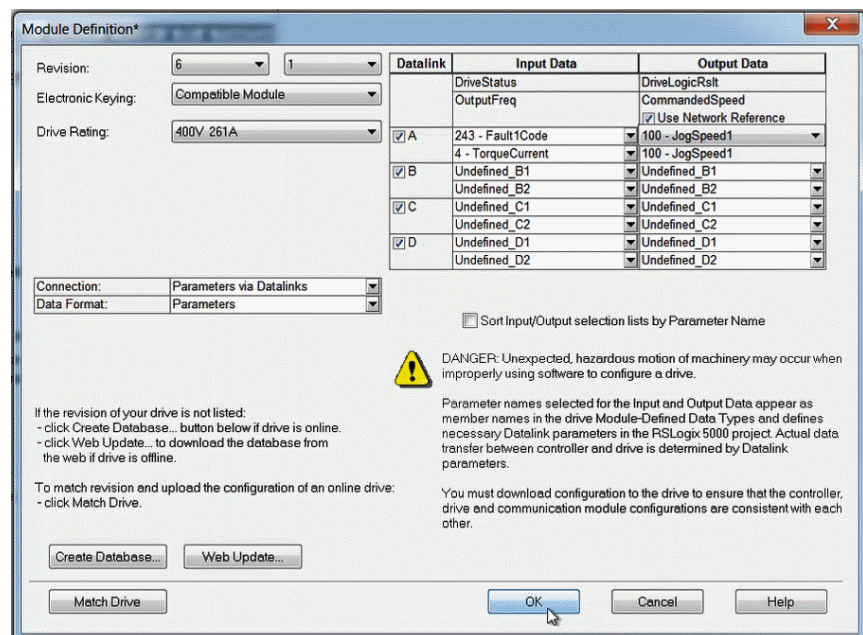
The required DataLink to add to your project are:

Input:

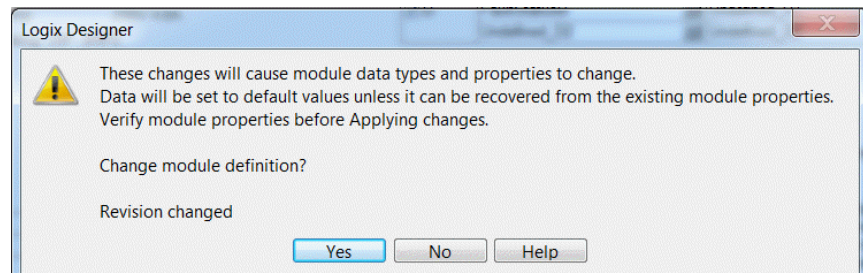
Fault1Code - 243

All Datalinks are optional

9. Once all Datalinks are provisioned click OK in the Module Definition dialog box.




10. Click YES in the confirmation dialog box to accept the changes to the datalinks.

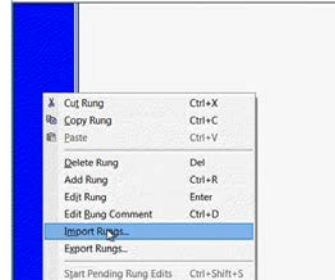


11. The selected device now appears in the project.

Add the Logic Rung

Follow these steps to import a rung into your project.

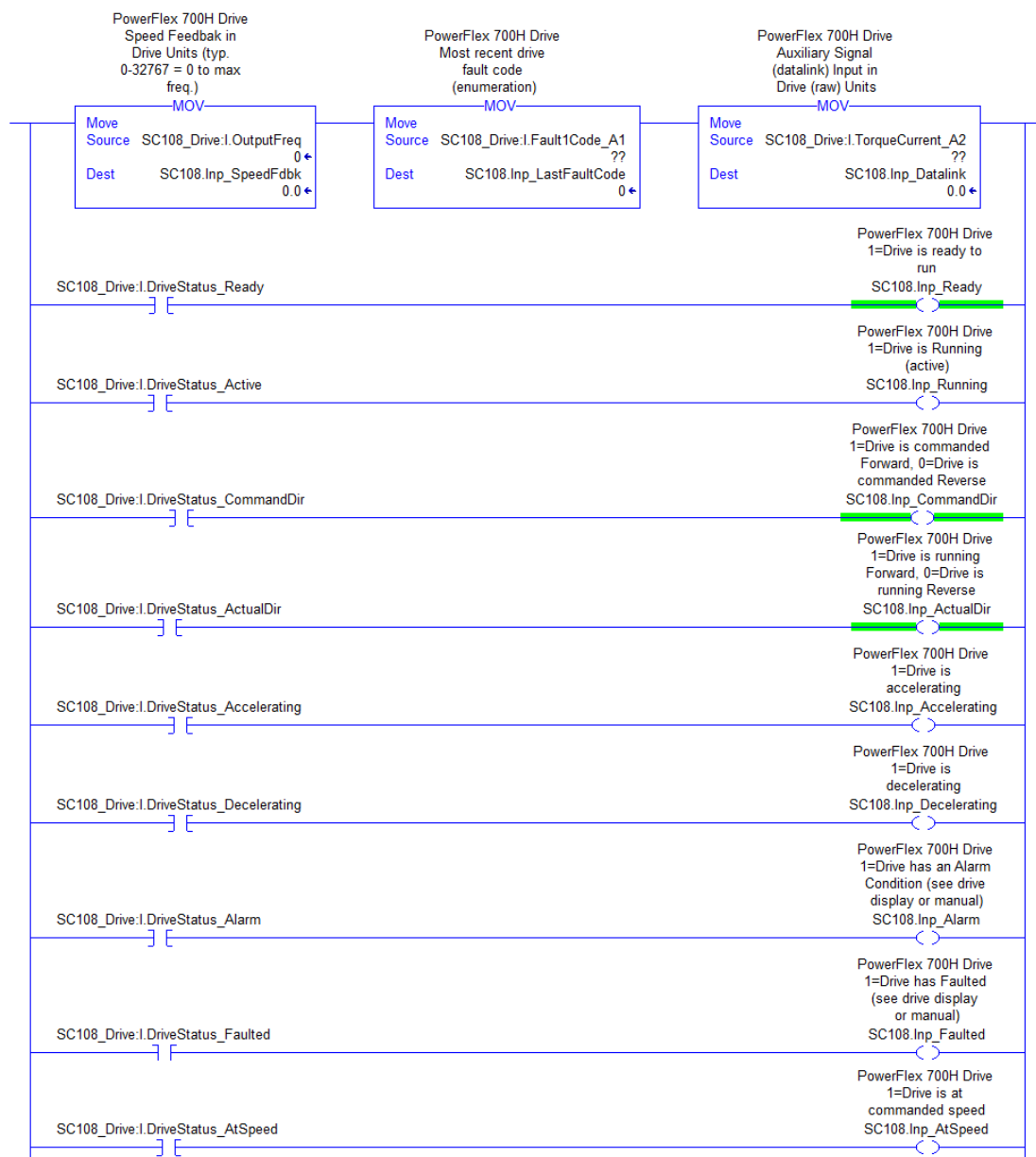
1. On the Controller Organizer, under Tasks, click  in front of Main Task.
2. Double-click Main_Routine to open this ladder logic routine.
3. Right-click one of the rungs and choose Import Rungs.



4. On the Import Rungs dialog box, select the device RUNG instruction and click Open.
5. During the import process, you can name the tags for the routine in the Import Configuration dialog box. In the Import Content tree, click Tags and type the names of the variables that match your process and the drive name in the Final Name column. Click OK when finished.
6. Click Browse (...) next to the GetFaultMSG tag to view the configuration.
7. Click the Communication tab and then Click Browse.
8. Set the message path to the device and Click OK.
9. Click OK in the Message Configuration dialog box.

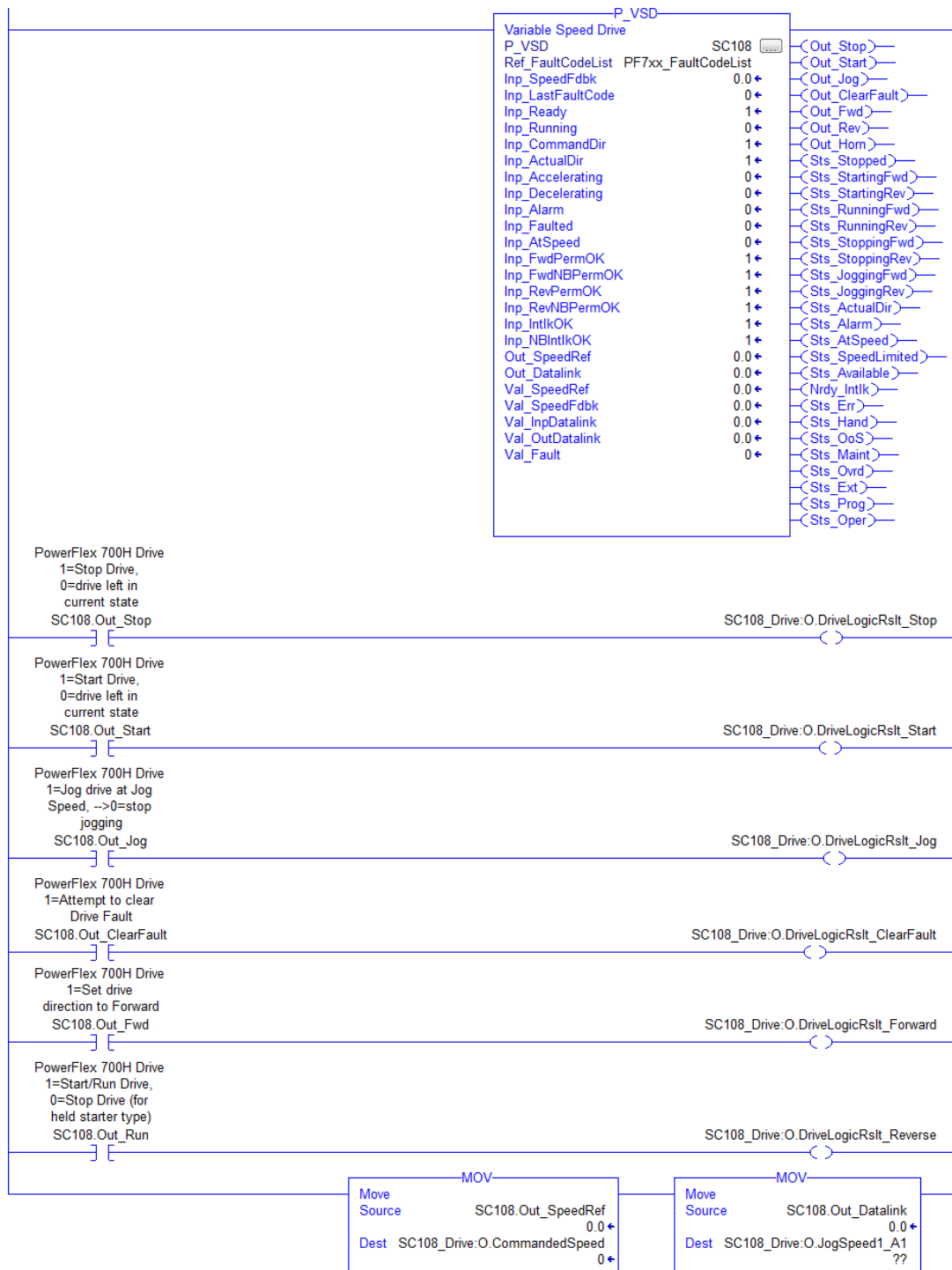
10. Map the Inputs of the P_VSD instruction to the device. These inputs are required:

- Inp_Ready
- Inp_Running
- Inp_CommandDir
- Inp_ActualDir
- Inp_Accelerating
- Inp_Decelerating
- Inp_Alarm
- Inp_Faulted
- Inp_AtSpeed



11. Map the outputs of the P_VSD instruction to the device. These outputs are required:

- Out_Stop
- Out_Start
- Out_Jog
- Out_ClearFault
- Out_Fwd
- Out_Rev





E1 Plus Electronic Overload Relay (P_E1PlusE)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_E1PlusE (E1 Plus™ Overload Relay (EtherNet/IP)) Add-On Instruction controls and monitors an E1 Plus Overload relay by using the 193-ETN EtherNet/IP interface module.

Functional Description

The E1 Plus Electronic Overload Relay (EtherNet/IP) instruction provides the following capabilities:

- Warning of impending overloads
- Identification of overload trip conditions
- Monitoring motor current as a percentage of full load amperes
- Monitoring percentage of thermal utilization (trip at 100%)
- Listing of last five trip causes (trip log)
- Configurable command to initiate a trip reset
- Monitoring of the states of the discrete inputs and discrete output of the relay.
- Monitoring of I/O communication faults
- Alarms for Trip Warning, Overload Trip, and I/O Fault
- Supports HMI 'breadcrumbs' for Alarm Inhibited, Bad Configuration, Not Ready

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_E1PlusE_4.10.00_AOIL5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

InOut Structure

InOut parameters are used to link the Add-On Instruction to external tags that contain necessary data for the instruction to operate. These external tags must be of the data type shown.

Tag Name	Data Type	Description
Inp	P_E1PlusE_Inp	E1PlusE Overload Parameter-based Input Assembly (100)

Operations

This section describes the primary operations for this Add-On Instruction.

Alarms

This instruction uses the following alarms, which are implemented by using embedded P_Alarm and P_Gate Add-On Instructions.

Alarm Name	P_Alarm Name	P-Gate Name	Description
I/O Fault	IOFault	None	Raised when the Inp_IOFault input is true. This input is used to indicate to the instruction that communication with the overload relay has failed. The device faceplate shows the I/O Source and Quality as communication failure flag a “Not Ready” diagnostic.
Overload Trip	Trip	None	Raised when the overload relay has tripped, helping prevent the motor from running. The overload relay must be reset before the motor can be started.
Pending Trip (Warning)	Warn	None	Raised when a motor overload condition is occurring and a trip of the overload relay is imminent. Immediate action must be taken to reduce the load on the motor.

Parameters of the P_Alarm object can be accessed by using the following convention: [P_Alarm Name].[P_Alarm Parameter].

Simulation

The P_E1PlusE Add-On Instruction does not have a Simulation capability.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	No EnableIn False logic is provided. Instruction parameters hold their last values.
Powerup (prescan, first scan)	Any commands received before first scan are discarded. Embedded P_Alarm instructions are handled in accordance with their standard power-up procedures. See the Reference Manual for the P_Alarm Instruction for more information.
Postscan (SFC transition)	No SFC postscan logic is provided.

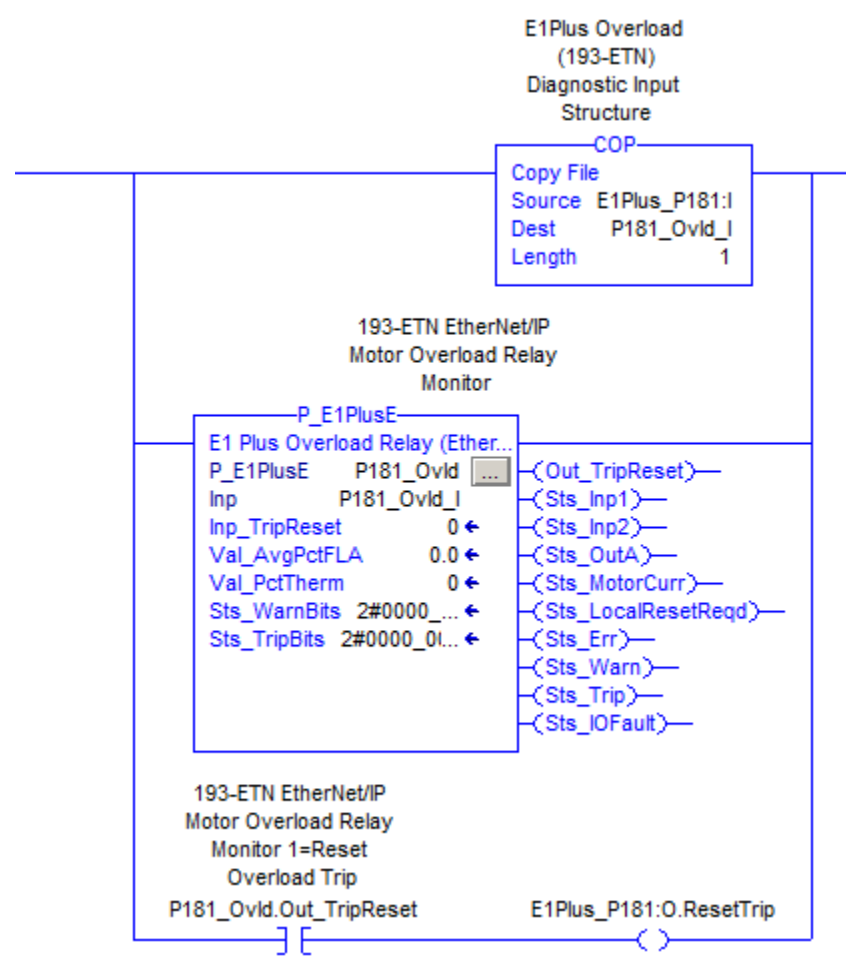
For more information, see the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#).

Programming Example

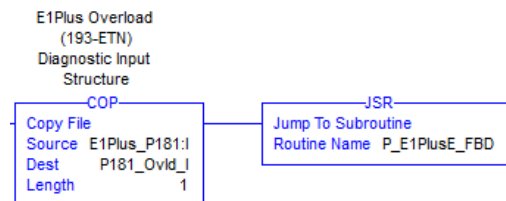
The following example shows the P_E1PlusE Add-On Instruction in both a strictly ladder and a combined ladder/function block context.

In both cases, ladder logic is used to copy the Module Defined Data Type for the E1 Plus module (AB:E1_Plus_Diag:I:0) to the User Defined Type for the E1Plus overload relay (catalog number 193-ETN) Diagnostic Input Structure (P_E1PlusE_Inp).

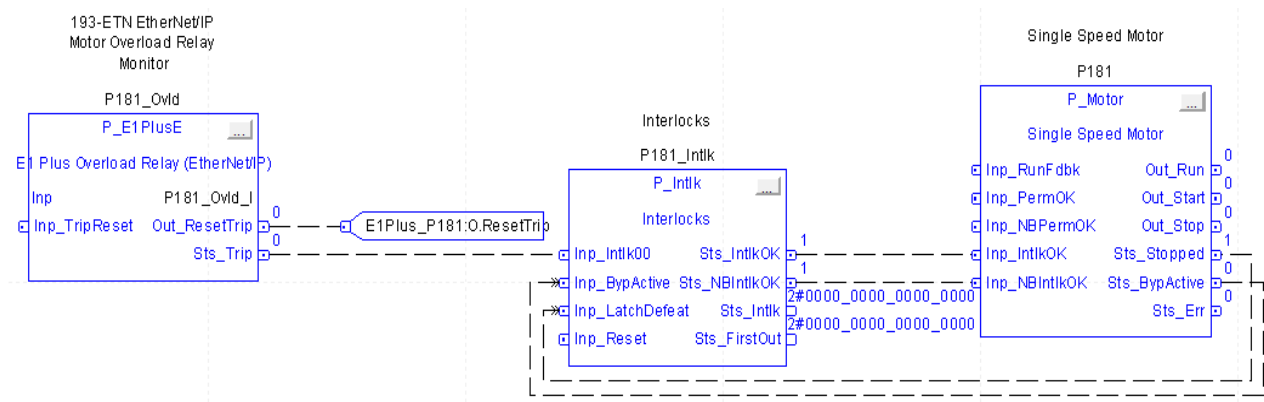
A complete ladder example is shown below.



An extended example using Function Blocks is also shown. In this case, the same COP instruction is used in ladder logic, followed by a Jump to Subroutine (JSR) to a Function Block routine.



The Function Block Routine shows a typical configuration with the P_E1PlusE Connected to an Interlock (P_Intlk) block followed by a Motor (P_Motor).



E3/E3Plus Overload Relay (P_E3Ovld)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_E3Ovld (E3™/E3 Plus™ overload relay) Add-On Instruction controls and monitors a 193/592-EC1, -EC2, -EC3, or -EC5 overload relay.

Functional Description

The E3/E3 Plus overload relay instruction provides the following capabilities:

- Warning of impending overloads
- Identification of overload trip conditions
- Countdown of time until overload trip can be reset
- Configurable command to initiate a remote test trip
- Configurable command to initiate a trip reset

TIP Three trips within a configurable time require resetting the trip locally (at the relay).

- Monitoring of the states of the discrete inputs and discrete outputs of the relay.
- Monitors input quality and communication status and provides value and indication of source and quality for the input
- Alarms for Trip Warning, Overload Trip, and I/O Fault
- Supports HMI 'breadcrumbs' for Alarm Inhibited, Bad Configuration, Not Ready

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_E3Ovld_4.10.00_AOIL5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Required Overload Configuration

Be sure that you configure the E3/E3 Plus parameters as follows:

- Input Assembly

The P_3Ovld instruction uses the Parameter-based Input Assembly. Set parameter 60 to 100 to select this assembly.

- Device Status Word (parameter 21) - set parameter 61 to 21
- Warning Status (parameter 15) - set parameter 62 to 15
- Trip Status (parameter 14) - set parameter 63 to 14
- Average Percent Full Load Amps (parameter 8) - set parameter 64...8

- Output Assembly

The P_3Ovld instruction uses Output Assembly 103 for E3 overload relays and Assembly 105 for E3Plus overload relays. Set parameter 59 to the appropriate assembly number, 103 or 105.

The P_E3Ovld Add-On Instruction uses only the Remote Trip and Remote Trip Reset bits in the Output assembly.

InOut Structure

InOut parameters are used to link the Add-On Instruction to external tags that contain necessary data for the instruction to operate. These external tags must be of the data type shown.

Tag Name	Data Type	Description
Inp	P_E3Ovld_Inp	E3 Overload Parameter-based Input Assembly (100)

Operations

This section describes the primary operations for this Add-On Instruction.

Alarms

This instruction uses the following alarms, which are implemented by using embedded P_Alarm Add-On Instructions.

Alarm Name	P_Alarm Name	P-Gate Name	Description
I/O Fault	IOFault	None	Raised when the Inp_IOFault input is true. This input is used to indicate to the instruction that communication with the overload relay has failed. The device faceplate shows the I/O Source and Quality as communication failure flag a 'Not Ready' diagnostic.
Overload Trip	Trip	None	Raised when the overload relay has tripped, helping prevent the motor from running. The overload relay must be reset before the motor can be started.
Pending Trip (warning)	Warn	None	Raised when a motor overload condition is occurring and a trip of the overload relay is imminent. Immediate action must be taken to reduce the load on the motor.

Parameters of the P_Alarm object can be accessed by using the following convention: [P_Alarm Name].[P_Alarm Parameter].

Simulation

The P_E3Ovld Add-On Instruction does not have a Simulation capability.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	No EnableIn False logic is provided. Instruction parameters hold their last values.
Powerup (prescan, first scan)	Any commands received before first scan are discarded. Embedded P_Alarm instructions are handled in accordance with their standard power-up procedures. See the Reference Manual for the P_Alarm Instruction for more information.
Postscan (SFC transition)	No SFC postscan logic is provided.

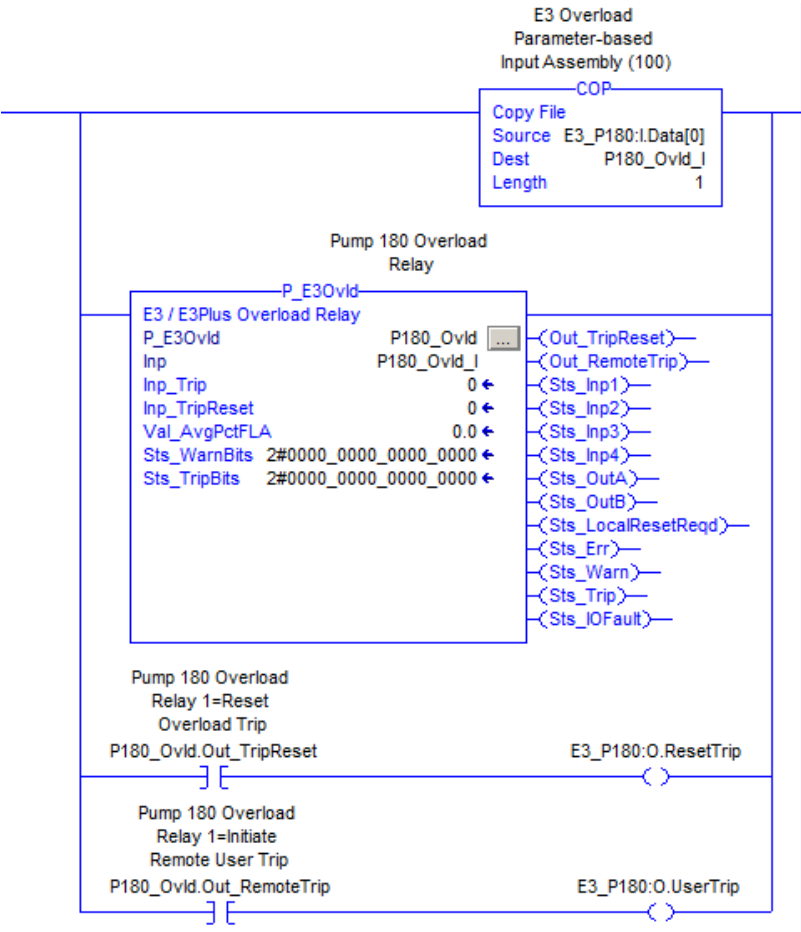
For more information, see the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#).

Programming Example

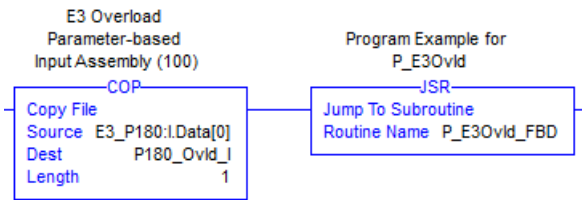
The following example shows the P_E3Ovld in both a strictly ladder and a combined ladder/function block context.

In both cases, ladder logic is used to copy the Module Defined Data Type for the E3 Overload Module (AB:E3 Plus_ParamBased_Generic:I:0) to the User Defined Type for the E3 Overload Parameter-based Input Assembly (100) (P_E3Ovld_Inp).

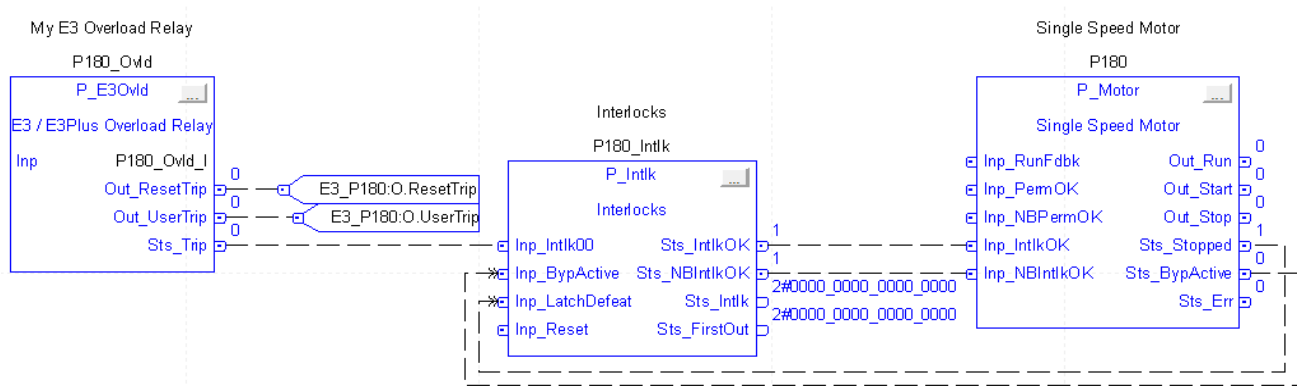
A complete ladder example is shown below.



An extended example using Function Blocks is also shown. In this case, the same COP instruction is used in ladder logic, followed by a Jump to Subroutine (JSR) to a Function Block routine.



The Function Block Routine shows a typical configuration with the P_E3Ovld Connected to an Interlock (P_Intlk) block followed by a Motor (P_Motor).



E300 Electronic Overload Relay (P_E300Ovld)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_E300Ovld (E300 Overload Relay) Add-On Instruction controls and monitors a 193-ECM-ETR (E300™ on EtherNet/IP) overload relay.

Functional Description

The P_E300Ovld (E300 Electronic Overload Relay) Add-On Instruction provides:

- Warning of impending overloads
- Identification of overload trip conditions
- Countdown of time until overload trip can be reset
- Configurable command to initiate a remote test trip
- Configurable command to initiate a trip reset

IMPORTANT Three trips within a configurable time require a reset of the trip locally (at the relay).

- Monitoring of states of the discrete inputs and discrete outputs of the relay
- Monitoring of various current, voltage, and ground fault values (if available)
- Monitoring of I/O communication faults
- Alarms for Trip Warning, Overload Trip, and I/O Fault
- Supports HMI 'breadcrumbs' for Alarm Inhibited, Bad Configuration, and Not Ready

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_E300Ovld_4.10.00_AOIL5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Required Overload Configuration

Be certain to configure the E300 Datalinks as follows.

IMPORTANT 'User Choice' Datalinks are not used by this Add-On Instruction and can be left unused or configured for your application.

- Input Assembly:
 - Datalinks:
 0. Time to Reset (Par 3)
 1. Time to Start (Par 31)
 2. Trip History #0 (Par 127)
 3. Warning History #0 (Par 133)
 4. Invalid Configuration Parameter Number (Par 38)
 5. User choice #1
 6. User choice #2
 7. User choice #3

- Output Assembly

The P_E300Ovld Add-On Instruction only uses the Remote Trip and Remote Trip Reset command bits in the output assembly.

InOut Structure

InOut parameters are used to link the Add-On Instruction to external tags that contain necessary data for the instruction to operate. These external tags must be of the data type shown.

Tag Name	Data Type	Description
Inp	P_E300Ovld_Inp	E300 Overload (193-ECM-ETR) Input Structure.
Ref_TripCodeList	P_DescList [*]	Tag containing List of Trip History Codes and Descriptions.
Ref_WarningCodeList	P_DescList [*]	Tag containing List of Warning History Codes and Descriptions.

The illustrations show the trip codes and warning codes list tags that are in each template.

These tags are pre-populated if you load the Add-On Instruction definition using the RUNG import or use a template or sample application that is included with the Library distribution.

Scope: ProcessObjects_4		Show: All Tags		Enter Name Filter...	
Name	Value	Force Mask	Style	Data Type	Description
+ E300_I.G.I	{...}	{...}		AB:E300:I0	
+ E300_I.G.O	{...}	{...}		AB:E300:O0	
+ E300_TripCodeList	{...}	{...}		P_DescList[26]	E300 Overload Relay Trip History Codes and Descriptions
+ E300_VIG.C	{...}	{...}		AB:E300:C0	
+ E300_VIG.I	{...}	{...}		AB:E300:I0	
+ E300_VIG.O	{...}	{...}		AB:E300:O0	
+ E300_WarningCodeList	{...}	{...}		P_DescList[24]	E300 Overload Relay Warning History Codes and Descriptions

Enter the tag name of the Trip Code list (E300_TripCodeList) in the Ref_TripCodeList parameter in each P_E300Ovld instruction instance.

The Trip Code list has preset codes and descriptions. You can use Trip Code list by copying the tag from a template application. You can also create your own by using the provided P_DescList data type.

+ E300_TripCodeList[0] Code	0	Decimal	DINT	Code / Description List Entry Code for which to look up Description
+ E300_TripCodeList[0] Desc	'No Fault Conditions, or Unkno...	{...}	STRING_40	Code / Description List Entry Description for given Code
- E300_TripCodeList[1]	{...}	{...}	P_DescList	E300 Overload Relay Trip History Codes and Descriptions Code / Description List ...
+ E300_TripCodeList[1] Code	1	Decimal	DINT	Code / Description List Entry Code for which to look up Description
+ E300_TripCodeList[1] Desc	'Motor Current Overload Trip'	{...}	STRING_40	Code / Description List Entry Description for given Code
- E300_TripCodeList[2]	{...}	{...}	P_DescList	E300 Overload Relay Trip History Codes and Descriptions Code / Description List ...
+ E300_TripCodeList[2] Code	2	Decimal	DINT	Code / Description List Entry Code for which to look up Description
+ E300_TripCodeList[2] Desc	'Motor Phase Loss Trip'	{...}	STRING_40	Code / Description List Entry Description for given Code
- E300_TripCodeList[3]	{...}	{...}	P_DescList	E300 Overload Relay Trip History Codes and Descriptions Code / Description List ...
+ E300_TripCodeList[3] Code	3	Decimal	DINT	Code / Description List Entry Code for which to look up Description
+ E300_TripCodeList[3] Desc	'Power Wire or Motor Winding G...	{...}	STRING_40	Code / Description List Entry Description for given Code

Enter the tagname of the Warning Code list (E300_WarningCodeList) in the Ref_WarningCodeList parameter of each P_E300Ovld instruction instance.

The Warning Code list has preset codes and descriptions. You can use Warning Code list by copying the tag from a template application. You can also create your own by using the provided P_DescList data type.

Scope: ProcessObjects_4		Show: All Tags				▼ Enter Name Filter...	
Name	Value	Force Mask	Style	Data Type	Description		
- E300_WarningCodeList	{...}	{...}		P_DescList(24)	E300 Overload Relay Warning History Codes and Descriptions		
- E300_WarningCodeList[0]	{...}	{...}		P_DescList	E300 Overload Relay Warning History Codes and Descriptions Code / Description ...		
+ E300_WarningCodeList[0] Code	0		Decimal	DINT	Code / Description List Entry Code for which to look up Description		
+ E300_WarningCodeList[0] Desc	'No Warning Conditions or Unkn...	{...}		STRING_40	Code / Description List Entry Description for given Code		
- E300_WarningCodeList[1]	{...}	{...}		P_DescList	E300 Overload Relay Warning History Codes and Descriptions Code / Description ...		
+ E300_WarningCodeList[1] Code	1		Decimal	DINT	Code / Description List Entry Code for which to look up Description		
+ E300_WarningCodeList[1] Desc	'Approaching Motor Current Ove...	{...}		STRING_40	Code / Description List Entry Description for given Code		
- E300_WarningCodeList[2]	{...}	{...}		P_DescList	E300 Overload Relay Warning History Codes and Descriptions Code / Description ...		
+ E300_WarningCodeList[2] Code	3		Decimal	DINT	Code / Description List Entry Code for which to look up Description		
+ E300_WarningCodeList[2] Desc	'Power Wire or Motor Winding G...	{...}		STRING_40	Code / Description List Entry Description for given Code		
- E300_WarningCodeList[3]	{...}	{...}		P_DescList	E300 Overload Relay Warning History Codes and Descriptions Code / Description ...		
+ E300_WarningCodeList[3] Code	5		Decimal	DINT	Code / Description List Entry Code for which to look up Description		
+ E300_WarningCodeList[3] Desc	'Motor Current Exceeds Jam War...	{...}		STRING_40	Code / Description List Entry Description for given Code		

Operations

This section describes the primary operations for this Add-On Instruction.

Alarms

This instruction uses the following alarms, which are implemented by using embedded P_Alarm Add-On Instructions.

Alarm Name	P_Alarm Name	P-Gate Name	Description
I/O Fault	IOFault	None	Raised when the Inp_IOFault input is true. This input is used to indicate to the instruction that communication with the overload relay has failed. The device faceplate shows the I/O Source and Quality as communication failure flag a “Not Ready” diagnostic.
Overload Trip	Trip	None	Raised when the overload relay has tripped, helping prevent the motor from running. The overload relay must be reset before the motor can be started.
Pending Trip (Warning)	Warn	None	Raised when a motor overload condition is occurring and a trip of the overload relay is imminent. Immediate action must be taken to reduce the load on the motor.

Parameters of the P_Alarm object can be accessed by using the following convention: [P_Alarm Name].[P_Alarm Parameter].

Simulation

The P_E300Ovld Add-On Instruction does not have a Simulation capability.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	No EnableIn False logic is provided. Instruction parameters hold their last values.
Powerup (prescan, first scan)	Any commands that are received before first scan are discarded. Embedded P_Alarm instructions are handled in accordance with their standard power-up procedures. See the Reference Manual for the P_Alarm Instruction for more information.
Postscan (SFC transition)	No SFC postscan logic is provided.

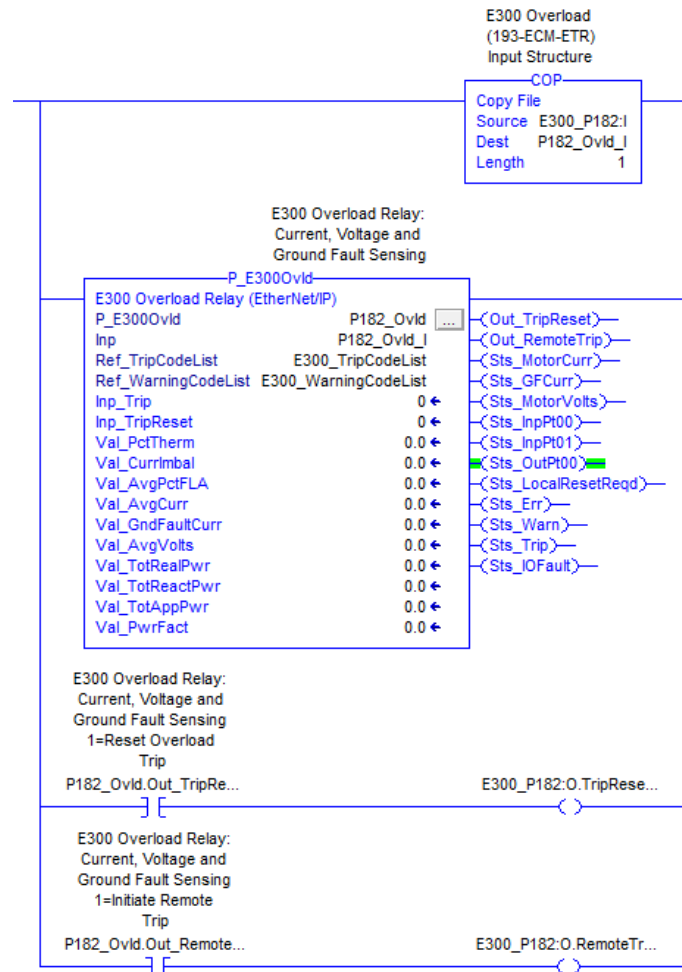
For more information, see the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#).

Programming Example

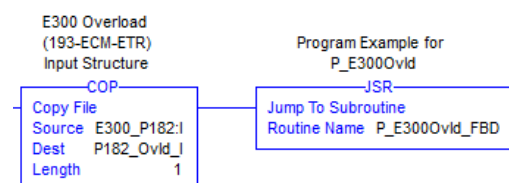
The following example shows the P_E300Ovld in strictly a ladder logic and a combined ladder/function block context.

In both cases, ladder logic is used to copy the Module Defined Data Type for the E300 Overload Module (AB:E300:I:0) to the user-defined data type for the E300 Overload (193-ECM-ETR) Input Structure (P_E300Ovld_Inp).

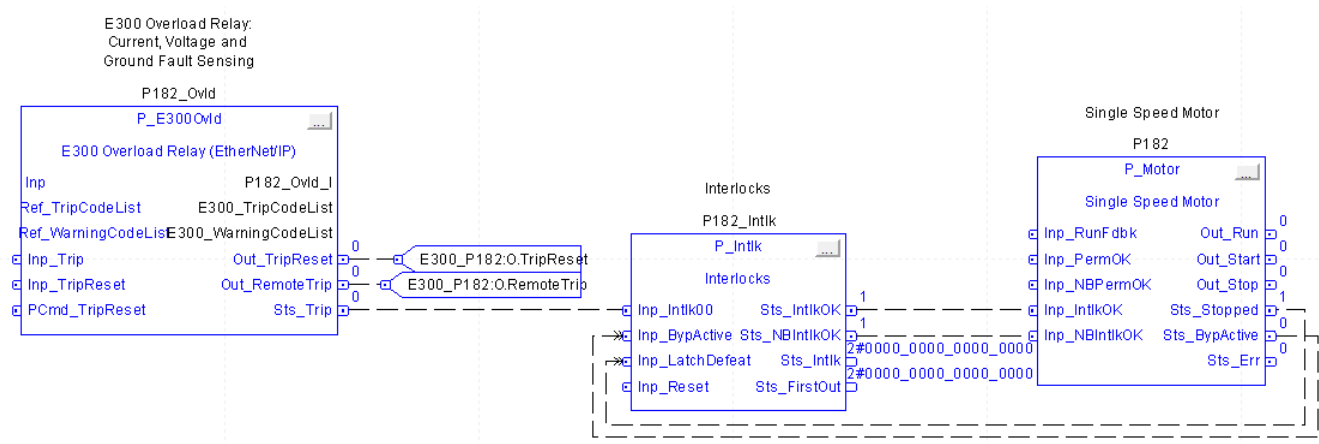
A complete ladder example is shown in the following figure.



An extended example using Function Blocks is also shown. In this case, the same COP instruction is used in ladder logic, followed by a Jump to Subroutine (JSR) to a Function Block routine.



The Function Block Routine shows a typical configuration with the P_E300Ovld connected to an interlock (P_Intlk) block followed by a motor (P_Motor).



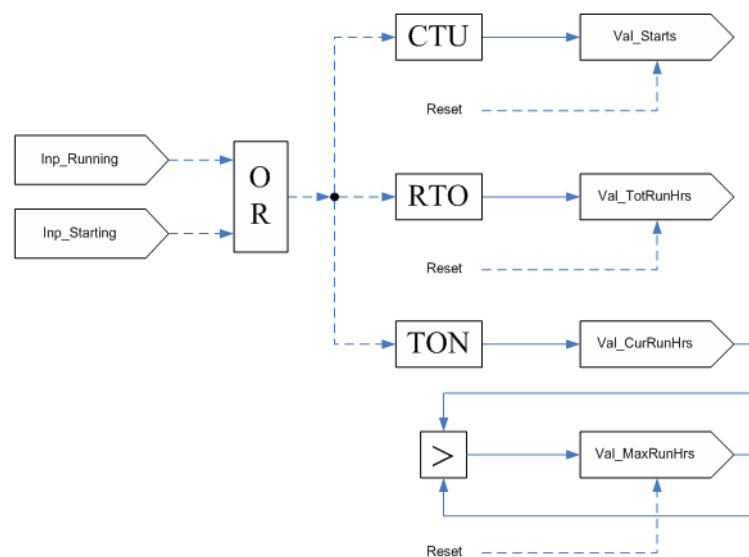
Runtime and Start Counter (P_RunTime)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_RunTime (Run Time and Start Counter) Add-On Instruction is used to accumulate the total runtime and count of starts for a motor or other equipment. It is a software implementation of the mechanical hour meter that is often mounted in the door of a motor control center (MCC) bucket to show total motor runtime. The runtime and number of starts are variables used by maintenance personnel to determine when to perform maintenance activities on the motor or other equipment.

Functional Description

The diagram shows the functional characteristics of this instruction.



The following list shows the functional coding.

Input	Starting	Running	Stopped
Inp_Starting	1	Ignored	0
Inp_Running	Ignored	1	0

The P_RunTime instruction provides the following capabilities:

- Accumulate and display the total running time for the associated equipment.
- Accumulate and display the count of starts or start attempts for the associated equipment.

- Show the amount of runtime since the last start (current run). This total is held after the equipment is stopped, until the next start, when it is reset to zero.
- Show the maximum amount of time for any single run; this is the highest value achieved by the previous total.
- Let maintenance personnel (but not operators) clear (individually) the total runtime, starts count, or maximum single runtime. This lets the times be reset when the motor or other equipment is serviced, rebuilt or replaced.

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_RunTime_4.10.00_AOIL5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (**boldfaced**) can change as service revisions are created.

Operations

This section describes the primary operations for this instruction.

Alarms

The P_RunTime Add-On Instruction provides no alarms. If alarms are required when any of its values exceed some limits, use one or more P_Alarm instructions with comparison logic. You can also use one or more P_AIn instructions to generate the necessary alarms.

Simulation

The P_RunTime Add-On Instruction does not have a Simulation capability.

Execution

The following table explains the handling of instruction execution conditions.

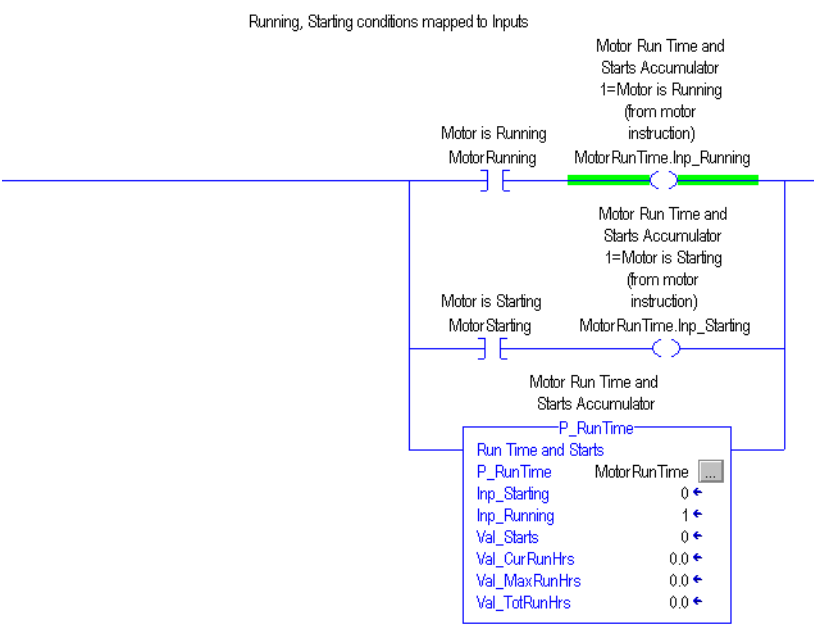
Condition	Description
EnableIn False (false rung) Handling	When executed with EnableIn false (a false rung), the runtime and Start Counter instruction is processed as if its timers and counter were on a false rung: timing stops, the Total runtime (retentive) is held, the Starts counter is set with its enable false (it counts when the rung goes true if the Running or Starting input is true) and the Current runtime (non-retentive) is reset. The last values of the instruction (outputs) are maintained while EnableIn is false. This processing lets the instruction be used as a ladder output instruction where the running state is provided as the rung condition. By setting the Running input (Inp_Running) to 1, the rung condition can be used to drive the instruction.
Powerup/Prescan Handling (initial modes)	On Powerup, the Current runtime is reset. Maximum Single runtime, Total runtime, and Starts Count are maintained.
Postscan (SFC transition) Handling	No SFC postscan logic is provided.

For more information, see the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#).

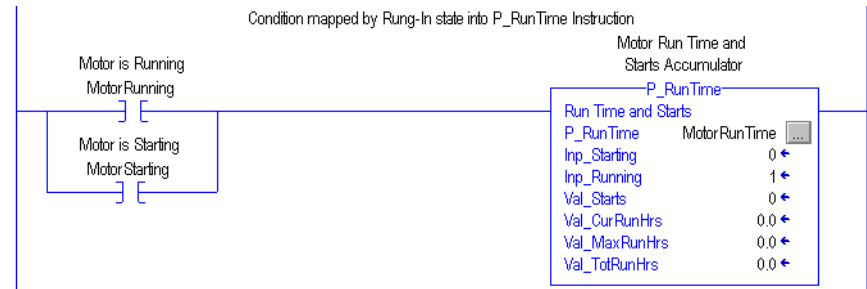
Implementation Using EnableIn False Feature

For the convenience of ladder diagram programmers, the P_RunTime instruction can be used in a ladder diagram routine with the input condition carried by the Rung-In condition instead of being mapped on a separate branch.

The following illustration shows normal implementation with the input conditions mapped on separate branches.



The following illustration shows EnableIn **false** implementation with the input condition mapped to the P_ResInh instruction using the Rung-In state.



The Rung-In condition determines whether the normal code ('Logic' Routine) for the Add-On Instruction is performed or its EnableIn False code ('EnableInFalse' Routine) is performed. In the P_RunTime instruction, the EnableIn False code performs the logic for a stopped motor. To use the Rung-In mapping method, Inp_Running must be set to 1 (its default value). When the rung is **true**, the logic executes for a running motor and runtime accumulates. When the rung is **false**, the logic executes for a stopped motor and runtime is not accumulated.

Restart Inhibit for Large Motor (P_ResInh)

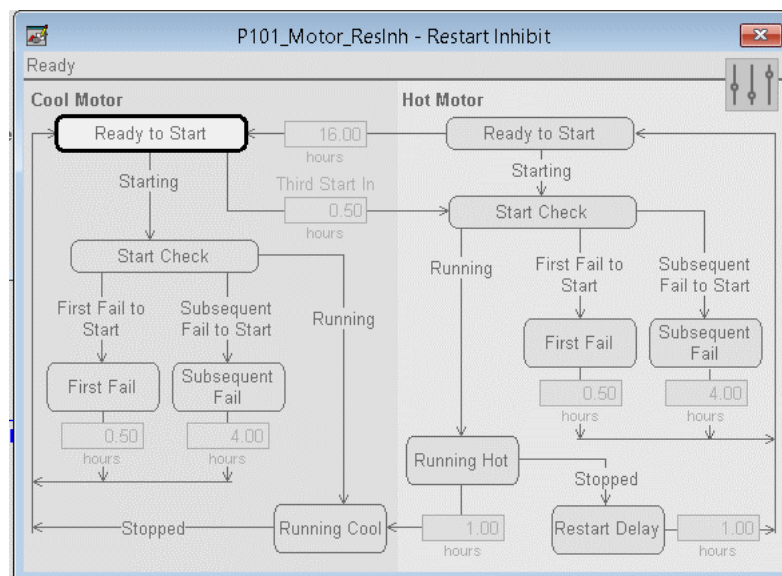
This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_ResInh (Restart Inhibit for Large Motor) Add-On Instruction is used to prevent damage to a large motor through repeated starts. The high starting current for a large motor causes considerable heating. Because the thermal mass of a large motor is much smaller relative to its horsepower and starting current compared to smaller motors, repeated starts (or start attempts) over a short time overheat the motor windings, potentially damaging the motor permanently.

The P_ResInh instruction provides a rule-based state model for restarts and is not intended to model or monitor the motor heating. It cannot replace sensor-based motor monitoring devices. It can, however, be a simple solution to avoid overstressing a motor without the cost (money or controller resources) of more extensive modeling and monitoring.

Functional Description

This screen capture shows the functional characteristics of this instruction.



The following list shows the functional coding.

Input	Starting	Running	Stopped
Inp_Starting	1	Ignored	Ignored
Inp_Running	0	1	Ignored
Inp_Stopped	Ignored	0	1

The P_ResInh instruction provides the following capabilities:

- Ready to Start signal for use by other logic when the motor can be started. Typically, this signal is used as a permissive in a motor-based control strategy.
- When the motor is not ready to start, provide a countdown of the time until the motor is ready to start (in minutes and seconds).

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix® firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_ResInh_4.10.00_AOIL5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Operations

This section describes the primary operations for this Add-On Instruction.

Alarms

The P_ResInh Add-On Instruction provides no alarms. It is typically used to provide a permissive condition for a motor or other equipment, not an interlock that requires an alarm.

Simulation

The P_ResInh Add-On Instruction does not have a Simulation capability. It uses the status of a related motor or drive object, which may be using its Simulation capability.

Execution

The following table explains the handling of instruction execution conditions.

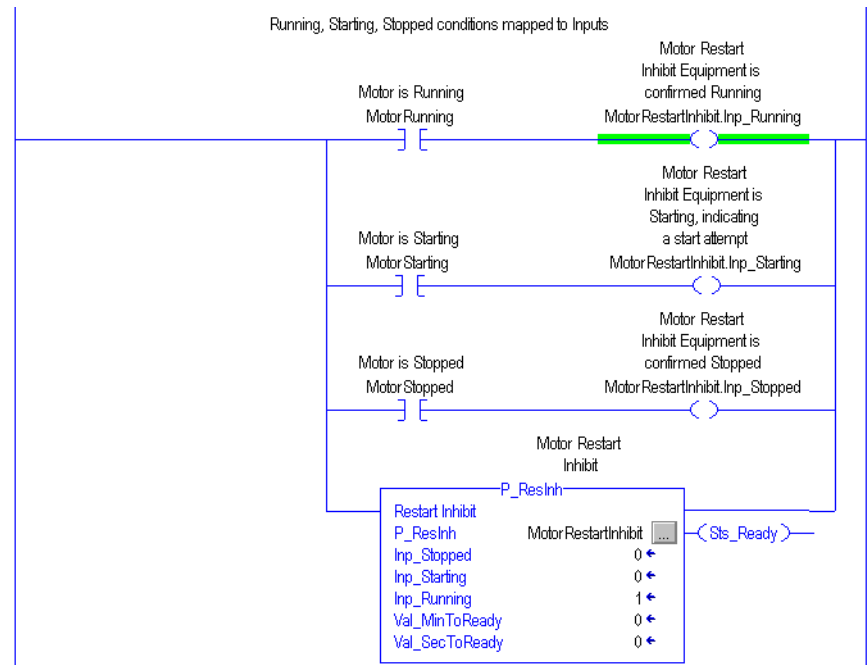
Condition	Description
EnableIn False (false rung)	Processing for EnableIn False (false rung) is handled as if the inputs indicate that the motor is stopped (Inp_Starting=0, Inp_Running =0, Inp_Stopped =1). This action lets the P_ResInh Add-On Instruction be coded on a ladder rung with an XIC of the motor running status. Inp_Running must be set to 1 and Inp_Starting must be set to 0 (their default values) for the instruction to be used in this fashion.
Powerup (prescan, first scan)	On prescan (Program to Run or Powerup), all timers are reset and the P_ResInh instruction reverts to the Cold Motor Stopped state if the motor is stopped, or the Cold Motor Running state if the motor is running.

For more information, see the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#).

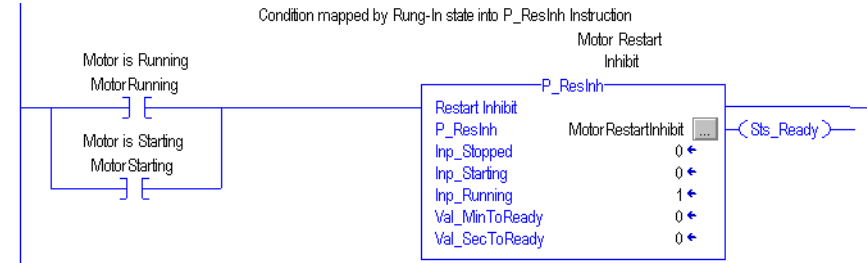
Implementation by Using EnableIn False Feature

For the convenience of ladder diagram programmers, the P_ResInh instruction can be used in a ladder diagram Routine with the input condition carried by the Rung-In condition instead of being mapped on separate branches.

The following illustration shows normal implementation with the input conditions mapped on separate branches.

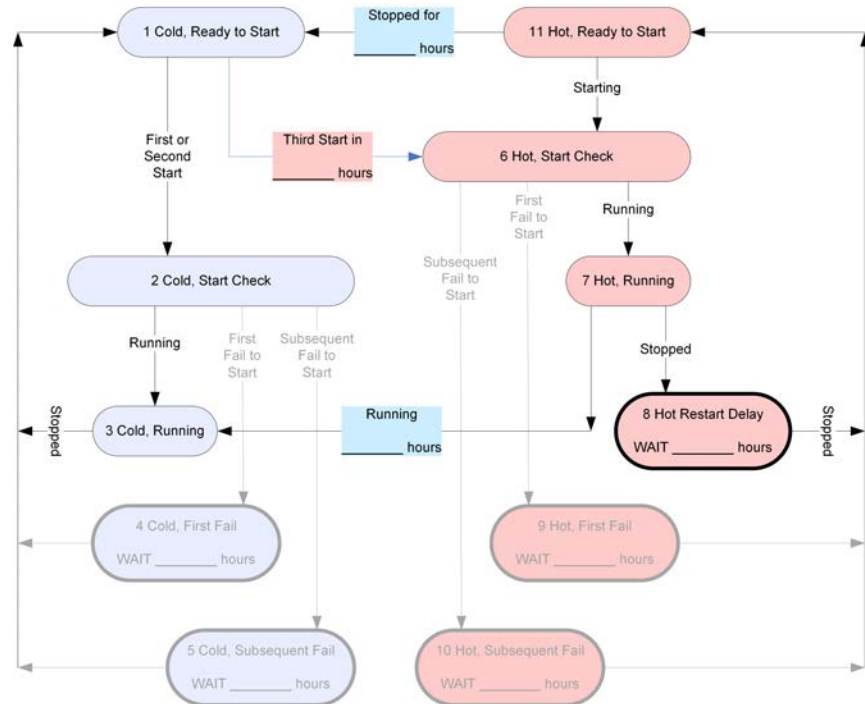


The following illustration shows the EnableIn False implementation with the input condition mapped to the P_ResInh instruction when the Rung-In state is used.



The Rung-In condition determines whether the normal code (logic routine) of the Add-On Instruction is executed or its EnableIn False code (EnableInFalse Routine) is executed. In the P_ResInh instruction, the EnableIn False code executes the logic for a stopped motor. Use the Rung-In mapping method, Inp_Running must be set to 1 (its default value). Then when the rung is true, the logic executes for a running motor. When the rung is false, the logic executes for a stopped motor.

The Starting Input (Inp_Starting) is not used, so the Fail to Start states are never reached. Effectively, the instruction uses the following state diagram when EnableIn False implementation is used.



Notes:

Valves

Purpose

This chapter is for the operation of the Add-on Instructions. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The Process Objects in this group provide an interface to a wide range of process valve types and valve statistical calculations.

[Table 17](#) describes the objects in this group, including when to use and not to use each one.

Table 17 - Valves

Process Object	Description	When to Use	When Not to Use
Analog/Pulsed Control Valve (P_ValveC)	This instruction manipulates a control valve by using an analog signal or discrete signals, and monitors the valve by using an analog position feedback. The valve requires an analog output (or analog value over a network) for the target position. Or, the valve requires a pair of discrete outputs (or discrete signals over a network). The outputs command the valve to move toward fully closed and when to move toward fully open.	<ul style="list-style-type: none"> • Want to use a control (modulating) valve where 0...100% is mapped to the percentage open. • Discrete outputs can be held (for a motor-operated control valve) or pulsed (for a ratcheting control valve). • Optionally, the valve provides, and you want to display, an actual position feedback signal. 	<ul style="list-style-type: none"> • You use one analog output control valve with no position feedback, with a PID loop (the P_PIDE Add-On Instruction or the PIDE built-in instruction). The CVEU signal from the PID block can be used directly to drive the valve. • You are using a two-state valve, such as a motor-operated or solenoid-operated valve that is driven only fully open or fully closed and not used to control flow or pressure. Use the appropriate two-state valve instruction. • Use P_ValveMO for motor operated valve (also for a dual solenoid-operated valve). • Use P_ValveSO for a single solenoid operated valve.

Table 17 - Valves

Process Object	Description	When to Use	When Not to Use
Hand-operated Valve (P_ValveHO)	<p>This instruction monitors a hand (locally) operated valve and displays its current state. The valve can have any type of actuator – handwheel, lever, motor, solenoid, pneumatic, hydraulic, and so on. But, the actuator is normally operated at the valve and only monitored by the control system via open and closed limit switches.</p> <p>The instruction cannot control the valve to both open and closed positions, but provides an optional Trip output to command the valve to its default (fail) position. If the trip function is used, the P_ValveHO instruction checks to make sure that the valve reaches the configured trip position (open or closed) if a trip command is executed.</p>	<ul style="list-style-type: none"> You monitor a valve (open/close) that is primarily operated by hand. The valve must have at least one limit switch for position sensing. The valve can use, but does not require, an output to remotely trip the valve to a 'safe' (default) position. On a trip, this instruction checks that the valve reaches the configured trip position and alarms if it does not within a configured time. 	<ul style="list-style-type: none"> Need to open and close a valve remotely. This instruction monitors (and optionally trips) only a locally operated valve. Need to operate a single-solenoid spring-return valve (fail closed or fail open), use the P_ValveSO instruction. Need to operate a motor-operated valve or other valve that requires separate open and close outputs, use the P_ValveMO instruction. Need to operate a multi-solenoid valve such as a mix-proof valve, use the P_ValveMP instruction. Need to operate a throttling valve (continuously variable), use the P_AOut or P_ValveC instruction. Need to operate other types of valves, try the P_DOut, P_D4SD, or P_nPos instructions.
Motor-operated Valve (P_ValveMO)	<p>This instruction operates (opens and closes) a motor-operated valve, monitoring for fault conditions.</p> <p>The valve can have, but does not require, a limit switch feedback for the ends of travel. The valve cannot require an output to trigger a 'valve stop' function. For example: breaking a seal-in circuit on the valve operator to stop travel or switch the direction of travel.</p>	<ul style="list-style-type: none"> Need to operate a motor-operated valve or other valve that requires separate open and close outputs. 	<ul style="list-style-type: none"> Need to operate a single-solenoid spring-return valve (fail closed or fail open). Use the Solenoid-operated Valve (P_ValveSO) Add-On Instruction. Need to operate a multi-solenoid valve that has positions (such as CIP™) other than 'opened' and 'closed'. Use the Mix-Proof Valve (P_ValveMP) Add-On Instruction. Need to monitor a valve that is primarily operated by hand. The valve could support a 'trip' output to drive it to a 'safe' position. Use the Hand-operated Valve (P_ValveHO) Add-On Instruction. Need a throttling (continuously variable) valve. Use the P_AOut instruction, the P_ValveC instruction, or operate the valve directly from a PIDE or PID built-in instruction. For some valves, you can also use the P_DOut, P_D4SD, or P_nPos instructions.
Mix-proof Valve (P_ValveMP)	<p>This instruction controls one mix-proof valve in various states. The instruction checks position feedback inputs to verify that the valve reaches the commanded position. An alarm can be provided on failure to reach a target position.</p> <p>Use this instruction if you want to operate a discrete mix-proof valve.</p>	<ul style="list-style-type: none"> Instruction supports mix-proof valves with or without additional connections for cleaning (CIP, clean-in-place) or steaming (SIP, sanitize in place). 	<ul style="list-style-type: none"> Need some other type of open-close valve: <ul style="list-style-type: none"> Use the P_Valve SO instruction for one solenoid-operated valve (single output with spring return to fail position). Use the P_ValveMO instruction for a motor-operated valve, or for a dual solenoid valve (separate open and close outputs). Use the P_ValveHO instruction for a hand-operated valve, which is a valve that is monitored only, or that has one output for tripping the valve to its failure position. Need a throttling (continuously variable) valve. Use the P_AOut (analog output) or P_ValveC (control valve) instruction.

Table 17 - Valves

Process Object	Description	When to Use	When Not to Use
Advanced Mix-Proof Valve (P_ValveMPAdv)	This instruction control one mix-proof valve in various states. The instruction checks position feedback inputs to verify that the value reaches the commanded position as well as intermediate positions to avoid cross-contamination. An alarm can be provided on failure to reach a requested position. The instruction supports separate interlocks for the Open, Lower Seat, Upper Seat and Cavity (cleaning) operations, allowing certain functions to be blocked independently.	<ul style="list-style-type: none"> Use this instruction with any mix-proof valve that supports independent open, lower seat, upper seat and cavity operations. Connections for clean-in-place or steam/sanitize-in-place (CIP and SIP) are supported, as are pulsing of seats while cleaning. This instruction is preferred over the P_ValveMP instruction for new applications because of the support for independent function interlocks. 	<ul style="list-style-type: none"> The P_ValveMP instruction can continue to be used for existing, tested applications. Use the P_ValveSO instruction for single-output solenoid valves that simply open and close. Use the P_ValveMO instruction for motor-operated valves or dual solenoid valves that simply open and close. Use the P_ValveHO instruction for hand-operated valves, which are monitored only or have a single output for tripping to a failure position. Use P_ValveC for throttling (variable-position) valves that use an analog output or dual discrete open/close outputs with an analog position feedback to control valve position.
Solenoid-operated Valve (P_ValveSO)	This instruction operates (opens and closes) one solenoid-operated valve, monitoring for fault conditions. If the valve is so equipped, monitor open/close limit switch feedback to verify that the Solenoid valve is opened or closed. Whether the Solenoid valve has each of the feedback limit switches can be configured at the engineer level. Whether to use each of the feedback limit switches can be configured at the Maintenance level.	<ul style="list-style-type: none"> Need to operate a single-solenoid spring-return valve, either energize-to-open (fail closed) or energize-to-close (fail open). The valve can have, but does not require, limit switch feedback for either or both ends of travel. Provides alarm for Full Stall if the valve feedback indicates it did not move off the original position within a configured amount of time when commanded to the other position. Provide an alarm for Transit Stall if the valve feedback indicates that the valve moved from the original position but did not reach the target position within a configured amount of time. 	<ul style="list-style-type: none"> Need to operate a motor-operated valve or other valve that requires separate Open and Close outputs. Use the P_ValveMO instruction. The P_ValveMO instruction can also be used for some dual-solenoid valves. Need to operate a multi-solenoid valve that has positions (such as for SIP/CIP) other than 'opened' and 'closed'. Use the P_ValveMP Mix-proof Valve Add-On Instruction. Need to monitor a valve that is primarily operated by hand. The valve could support a 'trip' output to drive it to a 'safe' position. Use the P_ValveHO instruction. Need a throttling (continuously variable) valve. Use the P_AOut instruction, the P_ValveC instruction, or operate the valve directly from a PIDE or PID built-in instruction. For some valves, the P_DOut, P_D4SD, or the P_nPos instructions can be used.
2-state Valve Statistics (P_ValveStats)	This instruction monitors a 2-state (open and close) valve and records various statistics that are related to stroke times and stroke counts. The instruction is designed to work with the P_ValveSO, P_ValveMO, and P_ValveHO instructions. This instruction can also be used with the P_ValveMP instruction.	<ul style="list-style-type: none"> Want to maintain stroke time and stroke count data on a 2-state valve to aid in planning maintenance or diagnosing valve and actuator problems. Want an indication when a valve takes longer to stroke than a configured threshold time. Valve is not an 'intelligent' valve that maintains its own valve stroke time and count data. You do not have a more specialized valve monitoring software, which provides functionality above and beyond what the P_ValveStats instruction provides. 	<ul style="list-style-type: none"> Need a continuously variable valve (control valve, throttling valve). The P_ValveStats instruction works only with valves that have full open and full close actions. The P_ValveStats instruction is not suitable for use with the P_AOut or P_ValveC instructions. Need an intelligent valve or valve monitoring and maintenance software that provides the same or more functionality than the P_ValveStats instruction. The valve or software provides more data than the P_ValveStats instruction, including specialized algorithms that predict impending valve failure or schedule maintenance.

Table 17 - Valves

Process Object	Description	When to Use	When Not to Use
n-Position Device (P_nPos)	<p>This instruction controls a circular or linear discrete device with 2...8 positions. The instruction provides outputs to select an individual position and outputs to move toward increasing positions ('clockwise' for a circular device) or decreasing positions ('counterclockwise' for a circular device).</p> <p>For linear devices, the P_nPos instruction can be configured for the following:</p> <ul style="list-style-type: none"> Return to Position 1 on every move Approach the target position from the 'same side' on each move to improve position repeatability Move directly to the new position. <p>For circular devices, the P_nPos instruction can be configured for the following:</p> <ul style="list-style-type: none"> Move only 'clockwise' to increase positions. For example, 6, 7, 8, 1, 2... Move both directions by using the shortest move. For example, 'clockwise' from 6...1: 6 7, 8, 1, or 'Counterclockwise' from 2...7: 2, 1, 8, 7 	<ul style="list-style-type: none"> Want to control the position of a device with 2...8 discrete positions. Accepts commands for the individual positions, increase and decrease position commands, or indexing cylinder commands. Supports devices with a locking or sealing capability. It can unlock or unseal the device, move to the new position, then lock or seal in position. Accepts position feedback (usually proximity or limit switches) and can alarm on failure to reach a target position in a configured time. When the locking/sealing capability is used, a lock/seal feedback can be provided and lock/unlock checking can also be performed. 	<ul style="list-style-type: none"> Want to control a simple two-state valve or two-state or three-state motor. The P_ValveMO, P_ValveSO, P_ValveMP, P_Motor, P_Motor2Spd, or P_MotorRev instruction provides a better interface and better 'model' for such a device. Need a continuously variable position device. The P_nPos instruction works only with devices that have 2...8 discrete positions. For most final control elements that are used in process control, a P_AOut or P_ValveC instruction is a better choice. For high-speed motion control, such as with servo drives, use the Motion Control instruction set provided within the Logix controller firmware.

Analog/Pulsed Control Valve (P_ValveC)

This section is for the operation of the Add-on Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_ValveC (Analog/Pulsed Control Valve) Add-On Instruction manipulates a control valve by using an analog signal or discrete signals, and monitors the valve by using an analog position feedback. The global objects and faceplate that is shown below are examples of the graphical interface tools for this Add-On Instruction.

Functional Description

The P_ValveC Instruction provides the following capabilities:

- Program and operator entry of the target valve position (percentage of the valve opening).
- Ramping of the valve position to the entered target at a specified rate of change (percent per second).
- Scaling of the Position Feedback from the valve from raw (I/O card or network value) units to percent open.
- Monitoring of Interlock conditions. When an Interlock condition is not OK, the valve can be configured to hold its current position or shed to a configured Interlock position.
- Monitoring for I/O communication faults. When an I/O communication fault occurs, this instruction can be configured to alarm only, or to shed, either to hold the current position or go to the configured Interlock position.
- Capability for maintenance personnel to take the drive out of service.

IMPORTANT This capability is not a substitute for hard lockout/tagout (LOTO) procedures.

- Provides alarms for Interlock trip, I/O fault, or an actuator-declared fault.
- Provides full-open and full-closed status-based on user-specified ranges for analog position feedback, or based on optional valve travel limit switches.
- Provides an 'available' status when the control valve instruction is in Program command source and the valve is being controlled.

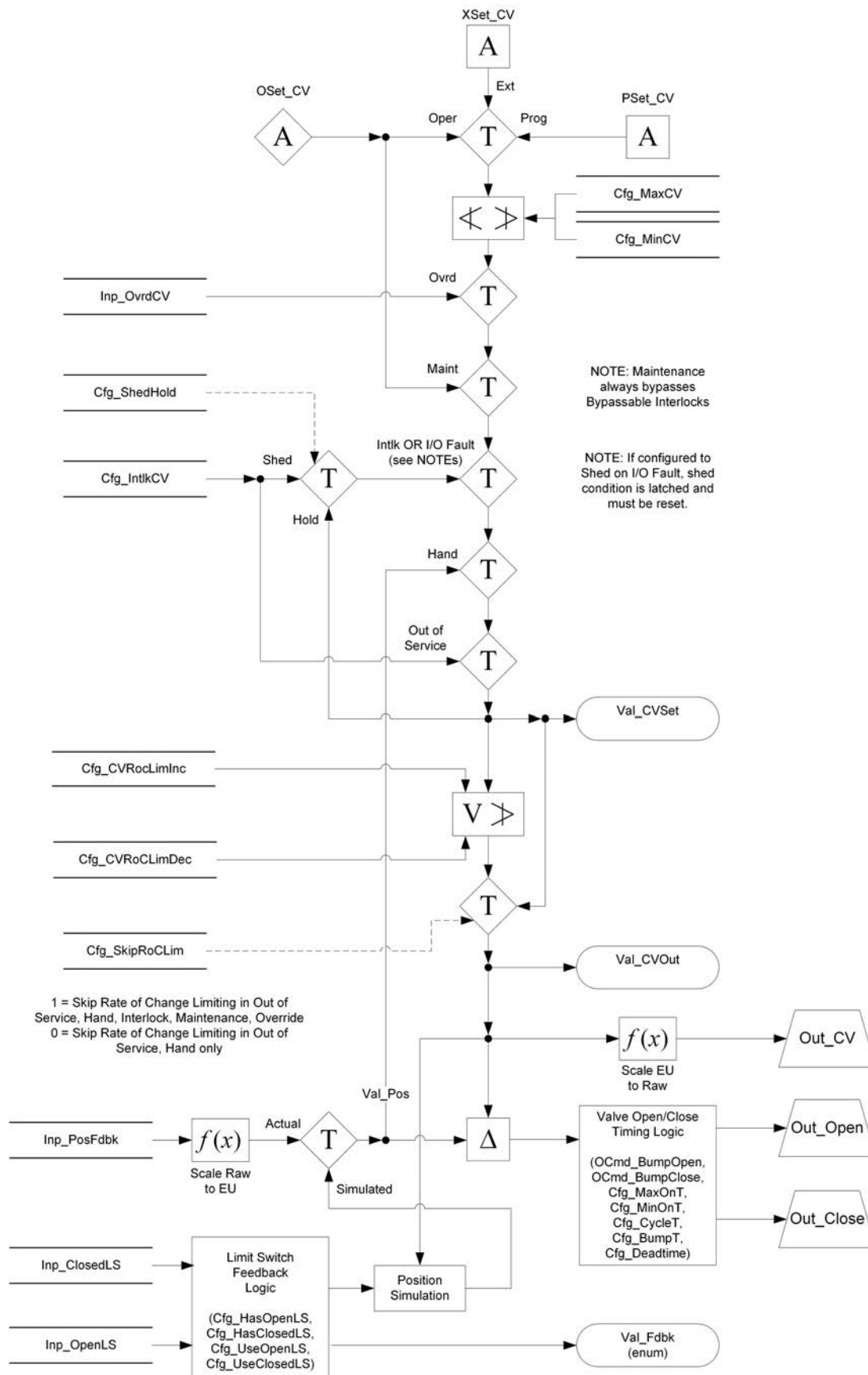
Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_ValveC_4.10.00_AOIL5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

The diagram shows the functional characteristics of the instruction.



Operations

This section describes the primary operations for this Add-On Instruction.

IMPORTANT See [Appendix A](#) for Command Sources and Simulation types.

Alarms


This instruction uses the following alarms, which are implemented by using embedded P_Alarm and P_Gate Add-On Instructions.

Alarm Name	P_Alarm Name	P_Gate Name	Description
Actuator Fault	ActuatorFault	None	Raised if the Inp_ActuatorFault input is true. This alarm is provided for use by valves that generate a fault contact, such as actuator motor overload trip.
I/O Fault	IOFault	None	Raised when the Inp_IOFault input is true. This input is used to indicate to the instruction that a communication failure has occurred for its I/O. If the I/O fault is configured as a shed fault, the output is set to the configure interlock CV or held at its last value until reset.
Interlock Trip	IntlkTrip	None	Raised when an Interlock 'not OK' condition causes the valve to move to the configured interlock position.

Parameters of the P_Alarm object can be accessed by using the following convention: [P_Alarm Name].[P_Alarm Parameter].

Simulation

Simulation in P_ValveC holds the analog output at zero and holds the discrete pulse outputs off while simulating a working valve. The I/O fault input is ignored.

You must set the Inp_Sim parameter in the Controller to '1' to enable simulation. The Simulation or Loopback Test icon  is displayed at the top left of the Operator faceplate, indicating the device is in simulation.

When you have finished in simulation, set the Inp_Sim parameter in the Controller to '0' to return to normal operation.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	Processing for EnableIn False (false rung) is handled the same as if the Control Valve were taken out of service by Command. The Control Valve outputs are de-energized and the Control Valve is shown as Program Out of Service on the HMI. All alarms are cleared.
Powerup (prescan, first scan)	The embedded P_CmdSrc and P_Alarm instructions handle the Processing of modes and alarms on pre-Scan and Powerup - refer to their specifications for details. On Power-up, the Control Valve command source is cleared and any commands that are received while the controller was in PROG command source are discarded. Otherwise, all data remains in the state it was in at power down.
Postscan	No SFC Postscan logic is provided.

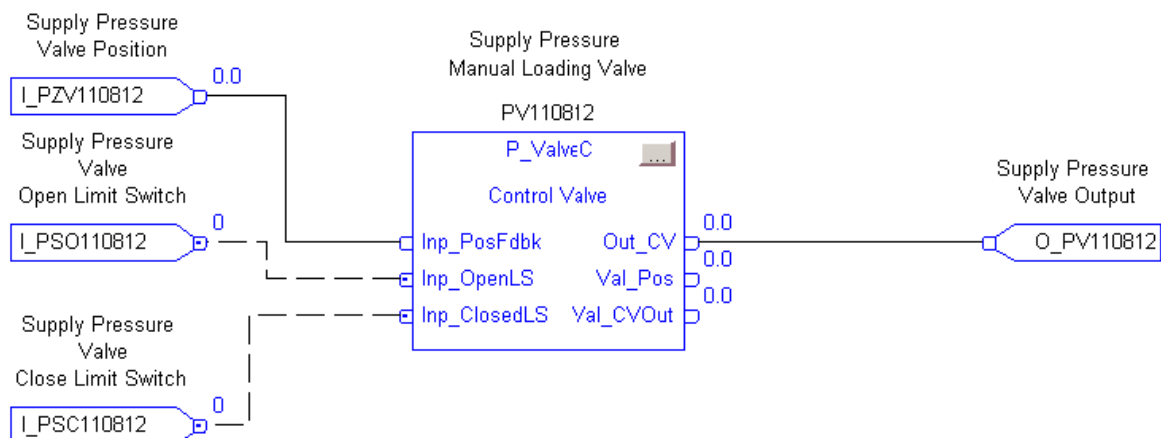
For more information, see the Logix 5000™ Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#).

Programming Examples

This section shows two examples for using the P_Valve C instruction.

Example 1: Manual Loading Station

This example uses the P_ValveC instruction to implement a manual loading station for a pressure control valve that is used to regulate gas supply to a process. The control valve in our example has open and closed limit switches and a position feedback. The desired valve position is provided by the operator through the HMI faceplate.



The field inputs for position feedback, open limit switch, and closed limit switch (as shown in the illustration) are connected to the instruction inputs Inp_PosFdbk, Inp_OpenLS, and Inp_ClosedLS. The Out_CV is connected to the field output going to the valve.

The parameters Cfg_HasOpenLS and Cfg_HasClosedLS are both set to 1 so the instruction knows that the field is providing open and closed limit switches. The parameters Cfg_UseOpenLS and Cfg_UseClosedLS are set to 1 so that these limit switches are used to determine device status.

The analog output card is expecting an output in units of 4...20 mA; however, the faceplate shows the value in terms of 0...100% open. Therefore, the scaling parameters are set as follows:

Cfg_CVEUMin:	0
Cfg_CVEUMax:	100
Cfg_CVRawMin:	4
Cfg_CVRawMax:	20

The feedback signal is also provided in units of 4...20 mA, so the parameters Cfg_FdbkRawMin and Cfg_FdbkRawMax are set to 4 and 20, respectively.

We want to limit the operator entry of the desired valve position to 80% open by setting Cfg_MaxCV to 80.

The parameters Cfg_HasIOFaultAlm, Cfg_HasActuatorFaultAlm, and Cfg_HasIntlkTripAlm are all set to 0 to indicate that no alarms are necessary for this device.

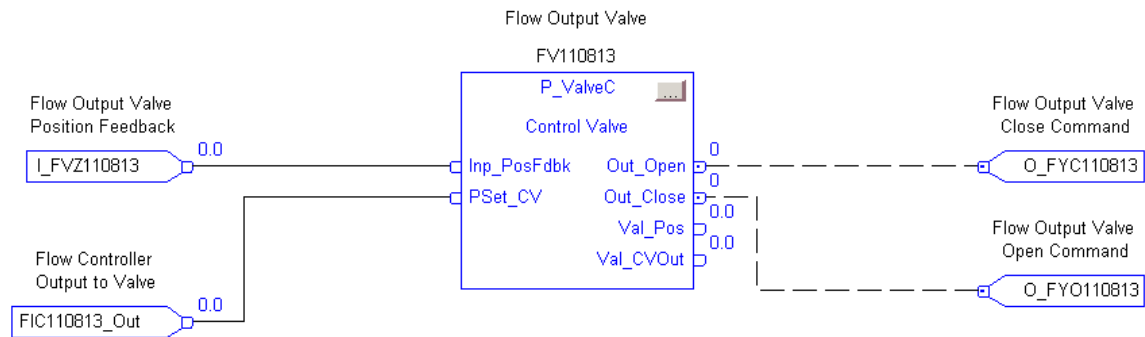
Lastly, configure the following local configuration tags to drive the text on the HMI faceplate. In this example, they are set as follows:

Cfg_Tag:	PV110812
Cfg_Label:	Gas Supply Valve
Cfg_Desc:	Gas Supply Valve Manual Loading Station
Cfg_EU:	%

Example 2: Ratcheting Control Valve

This example uses the P_ValveC instruction to automate a ratcheting valve that is driven open or closed by using two discrete outputs to control flow. The flow

valve in our example has a position feedback. The desired valve position is provided by an output of a control algorithm that is elsewhere in logic.



In this example, the field inputs for position feedback are wired (or connected) to the instruction input `Inp_PosFdbk`. `Out_Open` and `Out_Close` are connected to the field outputs going to the valve. The input to the instruction to set valve position is wired to `PSet_CV`. `Cfg_`

The instruction is normally operated by sequence program logic. The `P_CmdSrc` (command source) instruction within the `P_nPos` instance must also be configured. The parameter `CmdSrc.Cfg_ProgPwrUp` is set to 1 so the instruction will power up with the Program command source selected. The parameter `CmdSrc.Cfg_ProgNormal` is set to 1 to indicate that the instruction will normally use the Program command source. If another command source is selected, the graphic symbol and faceplate will flag the selected source.

The analog output is not used; however, the faceplate shows the value in terms of 0...100% open. Therefore, the scaling parameters are set as follows:

```
Cfg_CVEUMin:    0
Cfg_CVEUMAx:    100
Cfg_CVRawMin:    0 - default
Cfg_CVRawMax:    100 - default
```

The feedback signal is provided in units of 4...20 mA, so the parameters `Cfg_FdbkRawMin` and `Cfg_FdbkRawMax` are set to 4 and 20, respectively.

In this example, the ratcheting control valve is to be adjusted by cycling the open or close valve command for a period of time proportional to the amount the valve is to be moved. `Cfg_CycleT` is set to 10, to define the overall period of the cycle to cycle on and off the open or close output. `Cfg_OpenRate` and `Cfg_OpenCloseRate` are both set to 1, which means the required valve output is energized 1 second for every 1% difference between the desired position and the feedback position.

`Cfg_MaxOnT` is set to 5 so that the output is energized for no more than 5 seconds of the 10-second cycle time to allow for the valve to move, and the feedback to be verified before the next cycle. `Cfg_MinOnT` is set to 1 so that the output does not pulse if the calculated pulse time is less than 1 second.

The parameters Cfg_HasIOFaultAlm, Cfg_HasActuatorFaultAlm, and Cfg_HasIntlkTripAlm are all set to 0 to indicate that no alarms are necessary for this device.

Lastly, configure the following local configuration tags to drive the text on the HMI faceplate. In this example, they are set as follows:

Cfg_Tag:	FV110813
Cfg_Label:	Flow Control Valve
Cfg_Desc:	Flow Ratcheting Control Valve
Cfg_EU:	%

Hand-operated Valve (P_ValveHO)

This section is for the operation of the Add-on Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_ValveHO (Hand-operated Valve) Add-On Instruction monitors a hand (locally) operated valve and displays its current state. The valve may have any type of actuator – handwheel, lever, motor, solenoid, pneumatic, hydraulic – but it is normally operated at the valve and only monitored by the control system via open and closed limit switches. The P_ValveHO instruction cannot control the valve to both open and closed positions, but provides an optional Trip output to command the valve to its default (fail) position. If the trip function is used, the P_ValveHO instruction checks to make sure that the valve reaches the configured trip position (open or closed) if a trip command is executed.

Functional Description

The P_ValveHO instruction provides the following capabilities:

- Monitor the position feedback limit switch (or switches) from a valve and display actual valve status.
- Optionally can trip the valve (de-energize it or drive it to a default trip position). The program (via program commands) or the operator (via the HMI faceplate) can trip the valve any time.

The optional trip function provides the following capabilities:

- Detect failure to reach the configured trip position when tripped and generate an appropriate alarm.
- Monitor interlock conditions to trip the valve and alarm when an interlock initiates moving the valve to its trip position.
- Provide for simulation of a working valve while disabling the trip output, for use in off-process training, testing, or simulation.
- Monitor I/O communication, and alarm and trip if the shed on I/O fault function is enabled on a communication fault.

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix® firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_ValveHO_4.10.00_AOI.L5X Add-On instruction must be imported into the controller project to be used in the controller configuration. The service release number (**boldfaced**) can change as service revisions are created.

Operations

This section describes the primary operations for this Add-On Instruction.

Alarms


This instruction uses the following alarms, which are implemented by using embedded P_Alarm and P_Gate Add-On Instructions.

Alarm Name	P_Alarm Name	P_Gate Name	Description
Interlock Trip	IntlkTrip	None	Raised when the optional trip function is used and an interlock 'not OK' condition triggers the trip output to the valve. If interlocks are not bypassed, a bypassable interlock or a non-bypassable interlock 'not OK' condition initiates an interlock trip. If interlocks are bypassed, only a non-bypassable interlock 'not OK' condition initiates an interlock trip.
I/O Fault	IOFault	None	Raised when the Inp_IOFault input is true. This input is used to indicate to the instruction that a communication failure has occurred for its I/O. If the I/O Fault is configured as a shed fault and the optional trip function is used, the trip output is triggered until reset.
Transit Stall	TransitStall	None	Raised when the valve is using both open and closed limit switches and neither position is confirmed (the valve position is in transit) for the configured transit stall time.
Trip Failure	TripFail	None	Raised is the valve has and is using the optional trip feature, an attempt is made to trip the valve, and the limit switch feedbacks show that the valve did not reach the configured tripped position (opened or closed) within the configured fail to trip time.

Parameters of the P_Alarm object can be accessed by using the following convention: [P_Alarm Name].[P_Alarm Parameter].

Simulation

Simulation in P_ValveHO disables the normal input and lets you select a simulated input to see the reaction of the Hand-operated Valve.

You must set the Inp_Sim parameter in the Controller to '1' to enable simulation. The Simulation or Loopback Test icon  is displayed at the top left of the Operator faceplate, indicating the device is in simulation.

You can set the following parameters to simulate the corresponding input to the hand-operated valve:

- Inp_SimOpen - sets simulated valve state to open
- Inp_SimClose - sets simulated valve state to closed
- Cfg_SimFdbkT - number of seconds to wait for echo back of Open/Closed status when in Simulation

When you have finished in simulation, set the Inp_Sim parameter in the Controller to '0' to return to normal operation.

Execution

The following table explains the handling of instruction execution conditions.

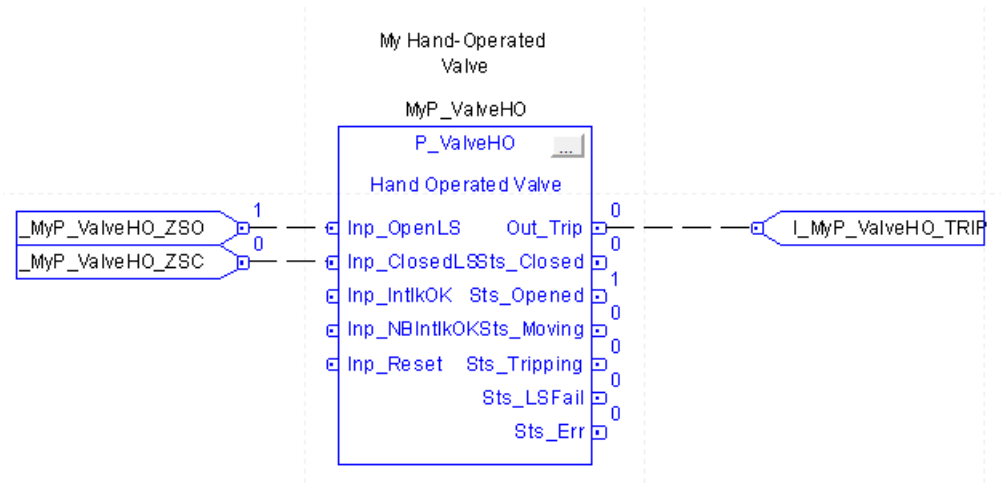
Condition	Description
EnableIn false (false rung)	Handled the same as if the trip function option were disabled. The trip output (Out_Trip) is de-energized if Cfg_HasTrip is 0, and is energized if Cfg_HasTrip = 1. All alarms are cleared.
Powerup (prescan, first scan)	Any commands that are received before first scan are discarded. The valve trip output is de-energized to help prevent a nuisance trip on first scan. Embedded P_Alarm instructions are handled in accordance with their standard powerup procedures. See the reference manual for the P_Alarm Instructions for more information.
Postscan (SFC transition)	No SFC postscan logic is provided.

For more information, see the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#).

Programming Example

The following is a simple example of P_ValveHO.

Boolean parameters I_MyP_ValveHO_ZSO and I_MyP_ValveHO_ZSC are used as inputs. A single output, I_MyP_ValveHO_TRIP is wired to an output that trips the valve to its fail position when energized.



Motor-operated Valve (P_ValveMO)

This section is for the operation of the Add-on Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_ValveMO (Motor-operated Valve) Add-On Instruction is used to operate (open and close) a motor-operated valve in various modes, monitoring for fault conditions.

Functional Description

The P_ValveMO instruction provides the following capabilities:

- Control of the Motor-operated Valve through the standard P_CmdSrc Add-On Instruction and modes.
- Ability to open or close a Motor-operated Valve, and if the valve is so equipped, monitor open/close limit switch feedback to verify the Motor-operated Valve is opened or closed. Whether the Motor-operated Valve has each of the feedback limit switches can be configured at the Engineer level. Whether to use each of the feedback limit switches can be configured at the Maintenance level.
- Optional ability to stop the motion of the Motor-operated Valve via a Stop Output, which is typically used to break the valve motor 'seal-in' circuit and stop the actuating motor. If the option to allow stopping the valve is enabled, the instruction lets the operator reverse travel. For example, select 'Open' while closing, which stops the valve, then moves it in the opposite direction.
- Alarm for Full Stall if the valve feedback indicates that it did not move off the original position within a configured amount of time when commanded to the other position. Also, an alarm for Transit Stall if the valve feedback indicates that the valve moved from the original position but did not reach the target position within a configured amount of time. The Transit Stall or Full Stall condition can optionally de-energize the outputs to the valve, requiring a reset.
- Limit switch failure indication if the limit switches indicate that the valve is not closed, not opened, and not moving (invalid state). The fail state, whether both switches are ON or both switches are OFF to indicate limit switch failure, is configurable.
- Open Permissives (those permissives that can be bypassed and those permissives that cannot be bypassed), which are conditions that allow the Motor-operated Valve to open. Close Permissives (those permissives that can be bypassed and those permissives that cannot be bypassed), which are conditions that allow the Motor-operated Valve to close. Maintenance personnel can bypass those permissives that can be bypassed.

- Capability for maintenance personnel to take the drive out of service.

IMPORTANT This capability is not a substitute for hard lockout/tagout (LOTO) procedures.

- Monitor an I/O fault input and alarm on an I/O fault. The I/O fault condition can optionally de-energize the outputs to the valve, requiring a reset.
- In Override command source, provide override inputs that determine whether the override is to open, close, or stop the motor-operated valve.
- A simulation capability, in which the outputs to the motor-operated valve are kept de-energized, but the object can be manipulated as if a working motor-operated valve were present. The delay between a command to open or close and the simulated opened or closed response is configurable. (This same delay is used if the motor-operated valve is configured with no open/close feedback.) This capability is often used for activities such as system testing and operator training.
- Actuator fault input for use by valves that generate a fault contact, such as actuator motor overload trip. The Actuator fault condition can optionally de-energize the outputs to the valve, requiring a reset.

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This code lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_ValveMO_4.10.00_AOIL5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Operations

This section describes the primary operations for this Add-On Instruction.

IMPORTANT See [Appendix A](#) for Command Sources and Simulation types.

Alarms

This instruction uses the following alarms, which are implemented by using embedded P_Alarm and P_Gate Add-On Instructions.

Alarm Name	P_Alarm Name	P_Gate Name	Description
Actuator Fault	ActuatorFault	None	Raised if the Inp_ActuatorFault input is true. This alarm is provided for use by valves that generate a fault contact, such as actuator motor overload trip. If the actuator fault is configured as a shed fault, the Stop output to the valve is triggered and a reset is required to command the valve open or closed.
Full Stall	FullStall	None	Raised when the valve has and is using Open and/or Closed limit switches, an attempt is made to open or close the valve, and the limit switches indicate that the valve did not move from its original position at all within the configured time.
I/O Fault	IOFault	None	Raised when the Inp_IOFault input is true. This input is used to indicate to the instruction that a communication failure has occurred for its I/O. If the I/O Fault is configured as a shed fault, the valve is commanded to Stop motion and cannot be commanded to either position until reset.
Transit Stall	TransitStall	None	Raised when the valve has and is using both open and closed position feedback, an attempt is made to open or close the valve, and the position feedback indicates that the valve moved off the original position but did not reach the target position within the configured transit stall time.

Parameters of the P_Alarm object can be accessed by using the following convention: [P_Alarm Name].[P_Alarm Parameter].

The Full Stall and Transit Stall Alarms have a configurable delay to allow the open and/or closed feedback time to align with the commanded output. This delay also provides time for the Motor-operated Valve to open or close.

Simulation

Simulation in P_ValveMO disables the outputs of the valve and simulates the feedback of a working valve.

You must set the Inp_Sim parameter in the Controller to '1' to enable simulation.

The Simulation or Loopback Test icon  is displayed at the top left of the Operator faceplate, indicating the device is in simulation.

While in simulation, you can set the number of seconds to wait (Cfg_SimFdbkT) before echoing back the Open/Closed status.

When you have finished in simulation, set the Inp_Sim parameter in the controller to '0' to return to normal operation.

Execution

The following table explains the handling of instruction execution conditions.

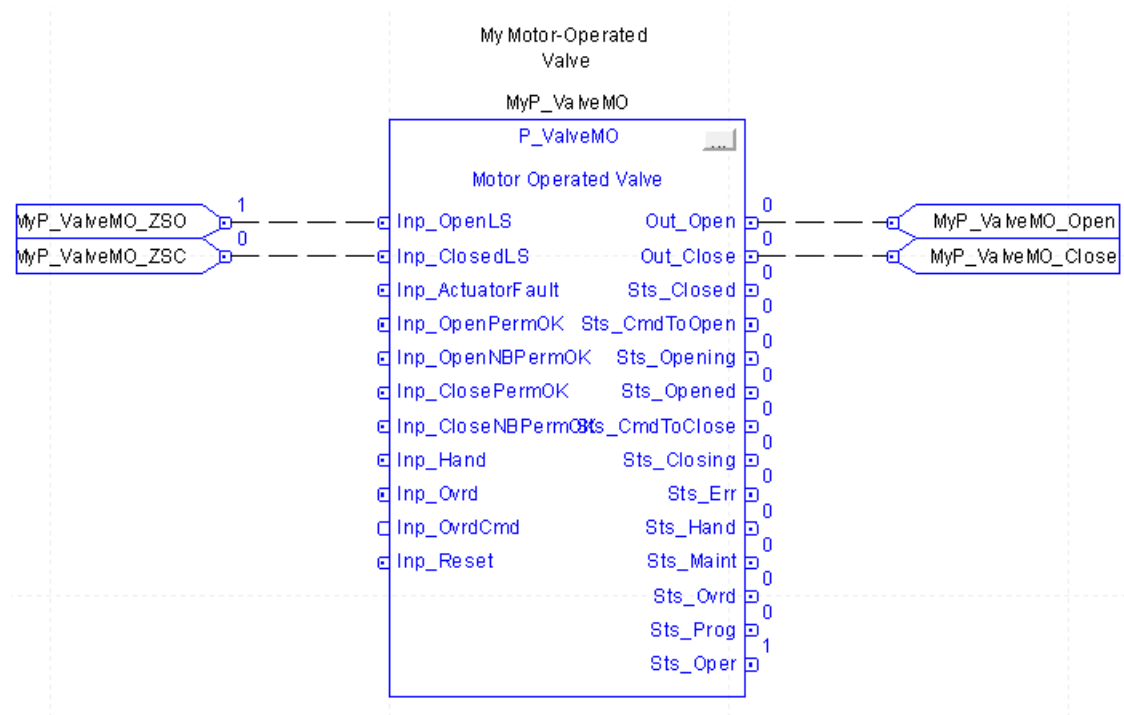
Condition	Description
EnableIn False (false rung)	Processing for EnableIn False (false rung) is handled the same as if the Motor-operated Valve were taken out of service by Command. The Motor-operated Valve outputs are de-energized and the Motor-operated Valve is shown as Program Out of Service on the HMI. All alarms are cleared.
Powerup (prescan, first scan)	The embedded P_CmdSrc and P_Alarm Add-On Instructions handle the processing of Modes and Alarms on prescan and powerup - refer to their specifications for details. On Powerup, the Motor-operated Valve is treated as if it had been commanded to stop motion.
Postscan (SFC transition)	No SFC postscan logic is provided.

For more information, see the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#).

Programming Example

The following is a simple example of P_ValveMO.

Boolean parameters MyP_ValveMO_ZSO and MyP_ValveMO_ZSC are used as limit switch feedback inputs. Two outputs, MyP_ValveMO_Open and MyP_ValveMO_Close, are wired to outputs that drive the valve open and closed.



Mix-proof Valve (P_ValveMP)

This section is for the operation of the Add-on Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The Mix-proof Valve (P_ValveMP) Add-On Instruction controls one mix-proof valve in a variety of modes and states, and can check position feedback inputs to verify that the valve reached the commanded position. An alarm can be provided on failure to reach a target position. The global objects and faceplate shown below are examples of the graphical interface tools for this Add-On Instruction.

Functional Description

The P_ValveMP Add-On Instruction provides the following capabilities:

- Operates a mix-proof valve with the following positions:
 - Closed
 - Opened
 - Lift upper seat (optional)*
 - Lift lower seat (optional)*
 - CIP/SIP leakage cavity (optional)
 - CIP/SIP upper seat (optional)*
 - CIP/SIP lower seat (optional)*

The asterisk (*) indicates that the position can pulse the seat being cleaned or lifted opened and closed to provide enhanced cleaning. (As the seat is popped open and closed, the flow velocity across the seat is increased compared to the fully open seat position.) Pulse times are configurable.

- Operated by using a state model that makes sure that valve seats are sequenced properly to avoid cross-contamination
- Provides six outputs and six inputs. The outputs in each valve state (including intermediate states) are configurable for on and off states. The inputs that verify each valve state are configurable for their required on, required off, and don't care states. Provides feedback checking to make sure that the valve reaches each position, including intermediate positions before moving to the next position. The time for feedback inputs to confirm that each state is configurable
- Graphic symbols are provided for mix-proof valves in 2-D layouts and 3-D (isometric) layouts for ease in building valve array and routing manifold displays
- Capability for maintenance personnel to take the drive out of service.

IMPORTANT This capability is not a substitute for hard lockout/tagout (LOTO) procedures.

- Provides inputs for Permissive conditions to enable moving the valve from the closed state
- Provides inputs for Interlock conditions to drive the valve to the closed state
- Monitors for I/O communication faults and closes the valve and alarms on a fault
- Provides an 'available' status for Program command source logic so automation code can know whether the valve can be controlled
- Provides a valve simulation capability. When the mix-proof valve is being simulated, outputs are left de-energized, and the instruction behaves as if a fully functioning valve were providing feedback

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_ValveMP_4.10.00_AOIL5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Operations

This section describes the primary operations for this Add-On Instruction.

IMPORTANT See [Appendix A](#) for Command Sources and Simulation types.

Alarms

This instruction uses the following alarms, which are implemented by using embedded P_Alarm and P_Gate Add-On Instructions.


Alarm Name	P_Alarm Name	P_Gate Name	Description
Fail	Fail	None	Raised when the valve is commanded to a new position and the device feedbacks fail to confirm that the valve reached each required position (see state diagram) within the configured time (Cfg_FailT). If the Failure is configured as a shed fault, the valve is commanded closed and cannot be opened until reset.

Alarm Name	P_Alarm Name	P_Gate Name	Description
Interlock Trip	IntlkTrip	None	Raised when an interlock 'not OK' condition causes the valve to transition from some other position to the closed position. If interlocks are not bypassed, a bypassable interlock or a non-bypassable interlock 'not OK' condition initiates an interlock trip. If interlocks are bypassed, only a non-bypassable interlock 'not OK' condition initiates an interlock trip.
I/O Fault	IOFault	None	Raised when an interlock 'not OK' condition causes the valve to transition from some other position to the closed position. If interlocks are not bypassed, a bypassable interlock or a non-bypassable interlock 'not OK' condition initiates an interlock trip. If interlocks are bypassed, only a non-bypassable interlock 'not OK' condition initiates an interlock trip.

Parameters of the P_Alarm object can be accessed by using the following convention: [P_Alarm Name].[P_Alarm Parameter].

Simulation

Simulation in P_ValveMP disables the normal outputs and provides the feedback of a working valve.

You must set the Inp_Sim parameter in the Controller to '1' to enable simulation. The Simulation or Loopback Test icon  is displayed at the top left of the Operator faceplate, indicating the device is in simulation.

When you have finished in simulation, set the Inp_Sim parameter in the controller to '0' to return to normal operation.

Execution

The following table explains the handling of instruction execution conditions.

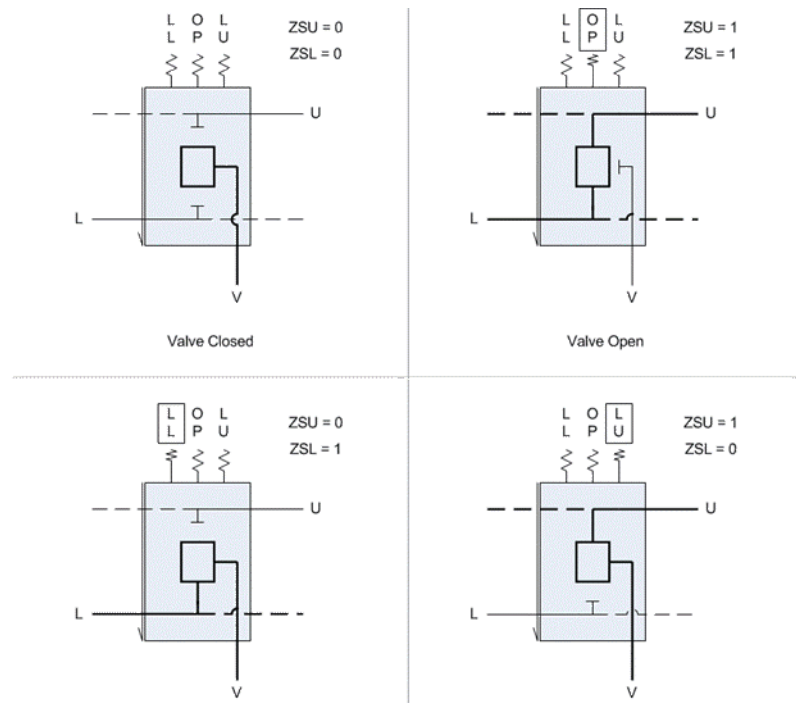
Condition	Description
EnableIn False (false rung)	Processing for EnableIn False (false rung) is handled the same as if the device were taken out of service by Command. The device outputs are de-energized and the device is shown as Program Out of Service on the HMI. All alarms are cleared.

Condition	Description
Powerup (prescan, first scan)	<p>On prescan, any commands that are received before first scan are discarded. The device is de-energized. On first scan, the device is commanded to the closed state.</p> <p>Embedded P_CmdSrc and P_Alarm instructions are handled in accordance with their standard powerup procedures.</p>
Postscan	No SFC Postscan logic is provided.

Programming Example

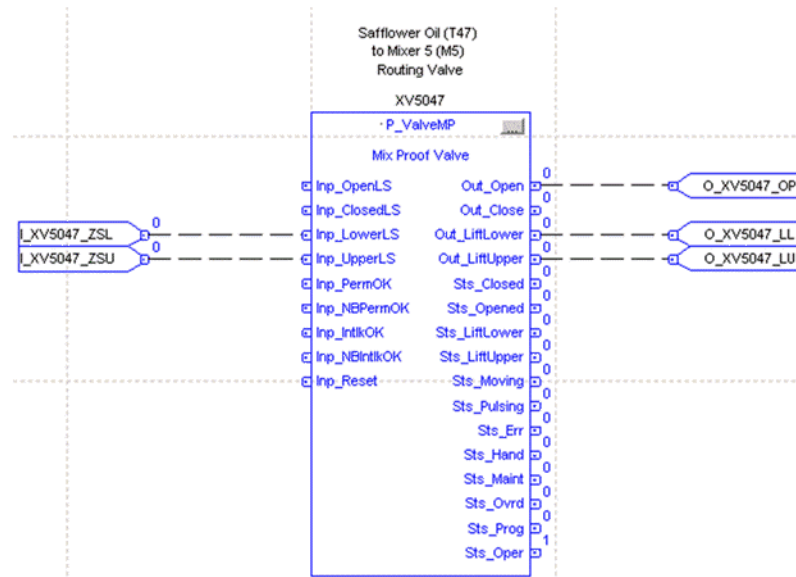
This example uses the P_ValveMP instruction to implement a mix-proof valve feeding bulk material (safflower oil) from a storage silo into a mixer.

For this example, the mix-proof valve connects to the control system by using two inputs and three outputs. The manufacturer's data sheet for the valve shows the following information.



In the closed position (all outputs off), flow is not directed anywhere and the valve cavity is vented to waste recovery. In the open state, flow is directed from the upper line to the lower line. In the Lower Lift position, flow is directed from the lower line through the valve cavity to waste recovery. In the Upper Lift position, flow is directed from the upper line through the valve cavity to waste recovery.

The P_ValveMP instruction for this valve is configured as shown in the following diagram.



The two limit switches from the field are connected into inputs Inp_LowerLS and Inp_UpperLS. The three outputs to the field are connected to Out_Open, Out_LiftLower, and Out_LiftUpper.

This programming example illustrates the remaining configuration by using the HMI faceplate. To use the faceplate to perform configuration, you must download the instruction above, have a display with a global object that is connected to this instruction instance, and have a FactoryTalk View software client with the appropriate security setup and access to this display. Each step of this programming example provides a listing of the configuration parameter values that were changed through the faceplate.

The descriptive strings and supported states are configured.

Parameter	Description
Cfg_Desc	Safflower Oil T-47 to Mixer M-5
Cfg_Label	Safflower T-47 / M-5
Cfg_Tag	XV-5047
Cfg_HasLiftLower, Cfg_HasLiftUpper	1 (checked)
Cfg_HasSIPCavity, Cfg_HasSIPLower, Cfg_HasSIPUpper	0 (unchecked)

The state configuration can be accessed and modified. For this example, we have to configure six states. There are three outputs only, so for each state, the outputs for Close, Cavity In, and Cavity Out can be ignored. We'll also ignore feedback states for feedback that is not required. The example uses parameters for all ignored values set to 0.

The table shows the configuration for each valve state.

State	Configuration
0 - De-energized	Cfg_OutStateTbl[0] = 2#0000_0000 (all outputs are 0) Cfg_FdbkReqdTbl[0] = 2#0000_1100 (Lift Lower and Lift Upper are required) Cfg_FdbkStateTbl[0] = 2#0000_0000 (Lift Lower and Lift Upper feedback state are 0)
1 - Close	Cfg_OutStateTbl[1] = 2#0000_0001 (Close output is 1 and all others are 0, Close output set for display purposes only) Cfg_FdbkReqdTbl[1] = 2#0000_1100 (Lift Lower and Lift Upper are required) Cfg_FdbkStateTbl[1] = 2#0000_0000 (Lift Lower and Lift Upper feedback state are 0)
3 - Open	Cfg_OutStateTbl[3] = 2#0000_0010 (Open output is 1 and all others are 0) Cfg_FdbkReqdTbl[3] = 2#0000_1100 (Lift Lower and Lift Upper are required) Cfg_FdbkStateTbl[3] = 2#0000_1100 (Lift Lower and Lift Upper feedback state are 1)
2 - Close Cavity Out	Cfg_OutStateTbl[3] = 2#0000_0010 (Open output is 1 and all others are 0) Cfg_FdbkReqdTbl[3] = 2#0000_1100 (Lift Lower and Lift Upper are required) Cfg_FdbkStateTbl[3] = 2#0000_1100 (Lift Lower and Lift Upper feedback state are 1)
4 - Lift Lower Seat	Cfg_OutStateTbl[4] = 2#0000_0100 (Lift Lower output is 1 and all others are 0) Cfg_FdbkReqdTbl[4] = 2#0000_1100 (Lift Lower and Lift Upper are required) Cfg_FdbkStateTbl[4] = 2#0000_0100 (Lift Lower feedback state is 1 and Lift Upper feedback state is 0)
5 - Lift Upper Seat	Cfg_OutStateTbl[5] = 2#0000_1000 (Lift Upper output is 1 and all others are 0) Cfg_FdbkReqdTbl[5] = 2#0000_1100 (Lift Lower and Lift Upper are required) Cfg_FdbkStateTbl[5] = 2#0000_1000 (Lift Lower feedback state is 0 and Lift Upper feedback state is 1)

From the faceplate, you can open the state configuration for all of the valve states simultaneously to compare and configure the states. On this message box, you set the states of the outputs, as well as the states of required feedback inputs, for the given valve state.

TIP You can open the state configuration message boxes for multiple states simultaneously. This can make it easier for you to check your selections against the documentation for your particular valve and actuator.

Advanced Mix-Proof Valve (P_ValveMPAdv)

This section is for the operation of the Add-on Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The Advanced Mix-proof Valve (P_ValveMPAdv) Add-On Instruction controls one mix-proof valve in a variety of modes and states, and can check position feedback inputs to verify that the valve reached the commanded position. An alarm can be provided on failure to reach a target position. The global objects and faceplate shown below are examples of the graphical interface tools for this Add-On Instruction.

Functional Description

The P_ValveMPAdv Add-On Instruction provides the following capabilities:

- Operates a mix-proof valve with the following positions:
 - Closed
 - Opened
 - Lift upper seat (optional)*
 - Lift lower seat (optional)*
 - CIP/SIP leakage cavity (optional)
 - CIP/SIP upper seat (optional)*
 - CIP/SIP lower seat (optional)*

The asterisk (*) indicates that the position can pulse the seat being cleaned or lifted open and closed to provide enhanced cleaning. (As the seat is popped open and closed, the flow velocity across the seat is increased compared to the fully open seat position.) Pulse times are configurable.

- Operated by using a state model that makes sure that valve seats are sequenced properly to avoid cross-contamination
- Provides six outputs and six inputs. The outputs in each valve state (including intermediate states) are configurable for on and off states. The inputs that verify each valve state are configurable for their required on, required off, and don't care states.
- Provides feedback checking to make sure that the valve reaches each position, including intermediate positions before moving to the next position. The time to wait after feedback inputs confirm the position before moving to the next valve position is configurable
- Graphic symbols are provided for mix-proof valves in 2-D layouts and 3-D (isometric) layouts for ease in building valve array and routing manifold displays
- Capability for maintenance personnel to take the valve out of service.

IMPORTANT This capability is not a substitute for hard lockout/tagout (LOTO) procedures.

- Unique to the P_ValveMPAdv instruction, provides inputs for Interlock conditions to drive the valve to the closed state. Separate interlock inputs are provided for:

- Valve open state
- Valve lower seat lift states
- Valve upper seat lift states
- Valve cavity / cleaning states

The separate interlock inputs allow an interlock to prevent lifting the lower seat or force the valve closed if the lower seat is lifted, for example, while still allowing the valve to open or lift the upper seat.

- Also unique to the P_ValveMPAdv instruction, provides for a valve locator beacon, either integral to the valve and actuator or mounted on or near the valve, so that the particular valve being controlled can be identified visually amongst a large number of valves in a piping matrix. The locator beacon also applies to the graphic symbol on the HMI, to aid the operator in finding the valve on screen.
- Monitors for I/O communication faults and closes the valve and raises an alarm on an I/O fault.
- Audible horn output to warn on commanded energize with configurable time to sound.
- Provides an 'available' status for Program command source logic so automation code can know whether the valve can be controlled.
- Provides a valve simulation capability. When the mix-proof valve is being simulated, outputs are left de-energized, and the instruction behaves as if a fully functioning valve were providing feedback.

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_ValveMPAdv_4.10.05_AOIL5X Add-On Instruction definition file must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created. The P_ValveMPAdv instruction is new for PlantPAx Library 4.10.05.

Operations

This section describes the primary operations for this Add-On Instruction.

IMPORTANT See [Appendix A](#) for Command Sources and Simulation types.

Alarms

This instruction uses the following alarms, which are implemented by using embedded P_Alarm and P_Gate Add-On Instructions.

Alarm Name	P_Alarm Name	P_Gate Name	Description
Fail	Fail	None	Raised when the valve is commanded to a new position and the device feedbacks fail to confirm that the valve reached each required position (see state diagram) within the configured time (Cfg_FailT). If the Failure is configured as a shed fault, the valve is commanded closed and cannot be opened until reset.
Interlock Trip	IntlkTrip	None	Raised when an interlock 'not OK' condition causes the valve to transition from some other position to the closed position. If interlocks are not bypassed, a bypassable interlock or a non-bypassable interlock 'not OK' condition initiates an interlock trip. If interlocks are bypassed, only a non-bypassable interlock 'not OK' condition initiates an interlock trip.
I/O Fault	IOFault	None	Raised when an interlock 'not OK' condition causes the valve to transition from some other position to the closed position. If interlocks are not bypassed, a bypassable interlock or a non-bypassable interlock 'not OK' condition initiates an interlock trip. If interlocks are bypassed, only a non-bypassable interlock 'not OK' condition initiates an interlock trip.

Parameters of the P_Alarm object can be accessed by using the following convention: [P_Alarm Name].[P_Alarm Parameter].

Simulation

Simulation in P_ValveMPAdv disables the normal outputs and provides the feedback of a working valve.

You must set the Inp_Sim parameter in the controller to '1' to enable simulation.

The Simulation or Loopback Test icon  is displayed at the top left of the Operator faceplate, indicating the device is in simulation.

When you have finished in simulation, set the Inp_Sim parameter in the controller to '0' to return to normal operation.

Execution

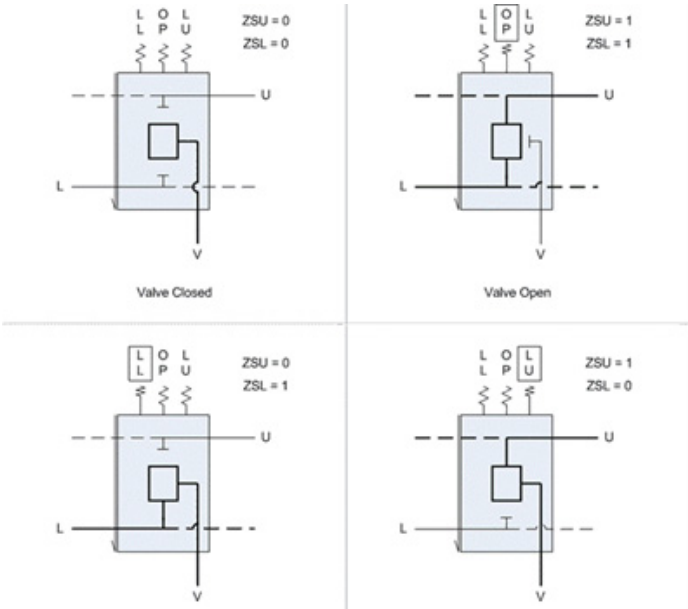
The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	Processing for EnableIn False (false rung) is handled the same as if the device were taken out of service by Command. The device outputs are de-energized and the device is shown as Program Out of Service on the HMI. All alarms are cleared.
Powerup (prescan, first scan)	On prescan, any commands that are received before first scan are discarded. The device is de-energized. On first scan, the device is commanded to the closed state. Embedded P_CmdSrc and P_Alarm instructions are handled in accordance with their standard powerup procedures.
Postscan	No SFC Postscan logic is provided.

Programming Example

This example uses the P_ValveMPAdv instruction to implement a mix-proof valve feeding bulk material (safflower oil) from a storage silo into a mixer.

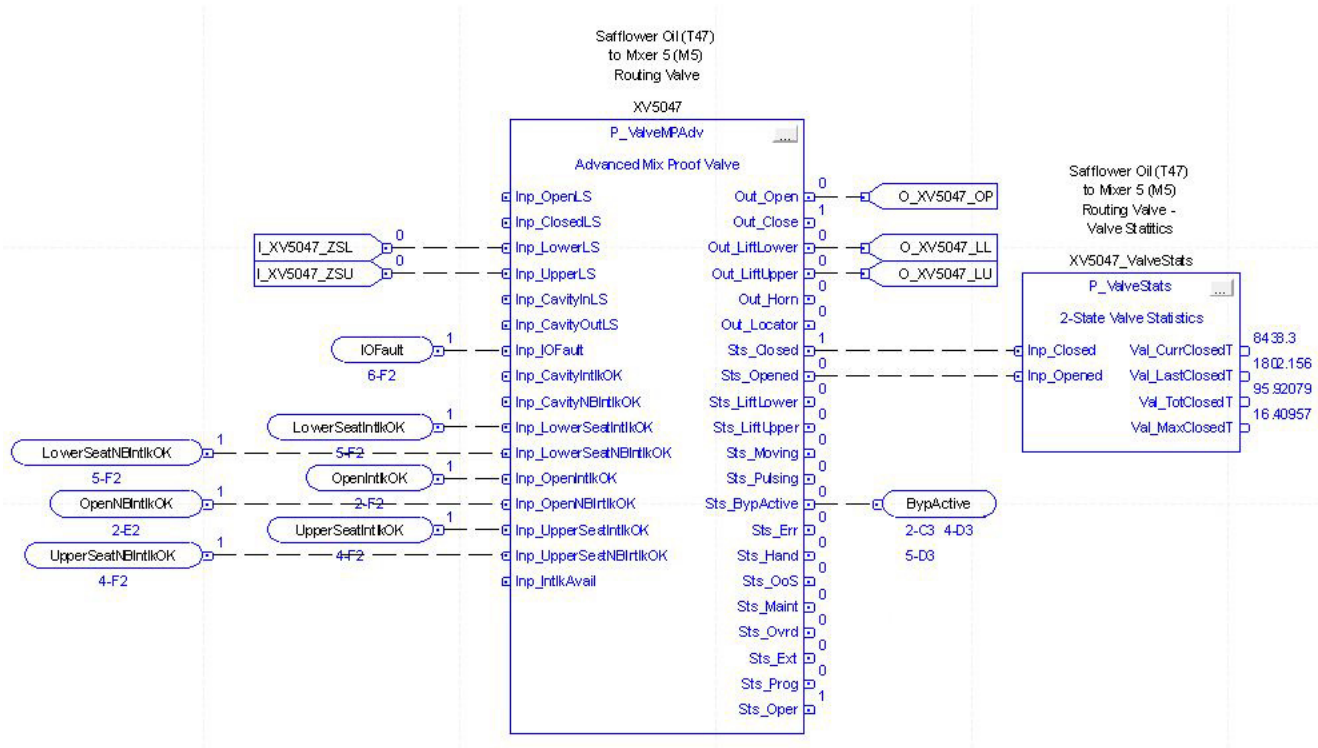
For this example, the mix-proof valve connects to the control system by using two inputs and three outputs. The manufacturer's data sheet for the valve shows the following information.



In the closed position (all outputs off), flow between the lower and upper lines is closed off, and any seat leakage to the central cavity is drained to waste recovery. In the open state, flow is directed from the upper line to the lower line. In the Lower Lift position, flow is directed from the lower line through the valve cavity to waste recovery. In the Upper Lift position, flow is directed from the upper line through the valve cavity to waste recovery.

Separate interlock conditions are used for the Open, Lower Lift and Upper Lift states, allowing interlocks to keep the valve from being commanded open while a seat lift cleaning operation is in progress.

The P_ValveMPAdv instruction for this valve is configured as shown in the following diagram.



The two limit switches from the field are connected into inputs Inp_LowerLS and Inp_UpperLS. The three outputs to the field are connected to Out_Open, Out_LiftLower, and Out_LiftUpper. There is an interlock instruction utilized for each output.

This programming example illustrates the remaining configuration by using the HMI faceplate. To use the faceplate to perform configuration, you must download the instruction above, have a display with a global object that is connected to this instruction instance, and have a FactoryTalk View software client with the appropriate security setup and access to this display. Each step of this programming example provides a listing of the configuration parameter values that were changed through the faceplate.

The descriptive strings and supported states are configured.

Parameter	Description
Cfg_Desc	Safflower Oil T-47 to Mixer M-5
Cfg_Label	Safflower T-47 / M-5
Cfg_Tag	XV-5047
Cfg_HasLiftLower, Cfg_HasLiftUpper	1 (checked)
Cfg_HasSIPCavity, Cfg_HasSIPLower, Cfg_HasSIPUpper	0 (unchecked)

The state configuration can be accessed and modified. For this example, we have to configure six states. There are three outputs only, so for each state, the outputs for Close, Cavity In, and Cavity Out can be ignored. We'll also ignore feedback states for feedback that is not required. The example uses parameters for all ignored values set to 0.

The table shows the configuration for each valve state.

State	Configuration
0 - De-energized	Cfg_OutStateTbl[0] = 2#0000_0000 (all outputs are 0) Cfg_FdbkReqdTbl[0] = 2#0000_1100 (Lift Lower and Lift Upper are required) Cfg_FdbkStateTbl[0] = 2#0000_0000 (Lift Lower and Lift Upper feedback state are 0)
1 - Close	Cfg_OutStateTbl[1] = 2#0000_0001 (Close output is 1 and all others are 0, Close output set for display purposes only) Cfg_FdbkReqdTbl[1] = 2#0000_1100 (Lift Lower and Lift Upper are required) Cfg_FdbkStateTbl[1] = 2#0000_0000 (Lift Lower and Lift Upper feedback state are 0)
3 - Open	Cfg_OutStateTbl[3] = 2#0000_0010 (Open output is 1 and all others are 0) Cfg_FdbkReqdTbl[3] = 2#0000_1100 (Lift Lower and Lift Upper are required) Cfg_FdbkStateTbl[3] = 2#0000_1100 (Lift Lower and Lift Upper feedback state are 1)
2 - Close Cavity Out	Cfg_OutStateTbl[3] = 2#0000_0010 (Open output is 1 and all others are 0) Cfg_FdbkReqdTbl[3] = 2#0000_1100 (Lift Lower and Lift Upper are required) Cfg_FdbkStateTbl[3] = 2#0000_1100 (Lift Lower and Lift Upper feedback state are 1)
4 - Lift Lower Seat	Cfg_OutStateTbl[4] = 2#0000_0100 (Lift Lower output is 1 and all others are 0) Cfg_FdbkReqdTbl[4] = 2#0000_1100 (Lift Lower and Lift Upper are required) Cfg_FdbkStateTbl[4] = 2#0000_0100 (Lift Lower feedback state is 1 and Lift Upper feedback state is 0)
5 - Lift Upper Seat	Cfg_OutStateTbl[5] = 2#0000_1000 (Lift Upper output is 1 and all others are 0) Cfg_FdbkReqdTbl[5] = 2#0000_1100 (Lift Lower and Lift Upper are required) Cfg_FdbkStateTbl[5] = 2#0000_1000 (Lift Lower feedback state is 0 and Lift Upper feedback state is 1)

From the faceplate, you can open the state configuration for all of the valve states simultaneously to compare and configure the states. On this message box, you set the states of the outputs, as well as the states of required feedback inputs, for the given valve state.

TIP You can open the state configuration message boxes for multiple states simultaneously. This can make it easier for you to check your selections against the documentation for your particular valve and actuator.

Solenoid-operated Valve (P_ValveSO)

This section is for the operation of the Add-on Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_ValveSO (Solenoid-operated Valve) Add-On Instruction is used to operate (open and close) a single solenoid-operated valve in various modes, monitoring for fault conditions.

Functional Description

The P_ValveSO instruction provides the following capabilities:

- Provide for control of the solenoid valve through the standard P_CmdSrc Add-On Instruction and modes.
- Provide for configuration of the de-energized state of the valve, Fail Open or Fail Closed (default), at the engineer level.
- Can open or close a solenoid valve. If the valve is so equipped, monitor open/close limit switch feedback to verify that the solenoid valve is opened or closed. Whether the solenoid valve has each of the feedback limit switches can be configured at the engineer level. Whether to use each of the feedback limit switches can be configured at the Maintenance level.
- An alarm for Full Stall if the valve feedback indicates it did not move off the original position within a configured amount of time when commanded to the other position. Provide an alarm for Transit Stall if the valve feedback indicates that the valve moved from the original position but did not reach the target position within a configured amount of time. The Transit Stall or Full Stall condition can optionally de-energize the output to the valve, requiring a reset.
- Provide a limit switch Failure indication if the limit switches indicate that the valve is not closed, not opened, and not moving. Provide a configuration for the failure state: whether both switches are ON or both switches are OFF to indicate limit switch failure.
- Provide for Permissives (those permissives that can be bypassed and those permissives that cannot be bypassed) which are conditions that allow the solenoid valve to energize. Also provide for Interlocks (those interlocks that can be bypassed and those interlocks that cannot be bypassed) which are conditions that de-energize the solenoid valve. Provide an Alarm when an Interlock de-energizes the solenoid valve. Provide Maintenance the capability to bypass the Permissives and Interlocks that can be bypassed.
- Capability for maintenance personnel to take the drive out of service.

IMPORTANT This capability is not a substitute for hard lockout/tagout (LOTO) procedures.

- Monitor an I/O Fault input and alarm on an I/O Fault. The I/O Fault condition can optionally de-energize the output to the valve, requiring a reset.
- In Override command source, provide an Override State input that determines whether the Override is to Open or Close the solenoid valve.
- Provide a Simulation capability, where the output to the solenoid valve is kept de-energized, but the instruction can be manipulated as if a working solenoid valve were present. The delay between a command to Open or Close and the simulated opened or closed response is configurable. (This same delay is used if the solenoid valve is configured with no Open/Close feedback.) This capability is often used for activities such as instruction testing and operator training.
- Provide an output suitable for holding the solenoid valve coil energized (to open or close, based on the configured fail state).

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This code lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_ValveSO_4.10.00_AOIL5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Operations

This section describes the primary operations for this Add-On Instruction.

IMPORTANT See [Appendix A](#) for Command Sources and Simulation types.

Alarms

This instruction uses the following alarms, which are implemented by using embedded P_Alarm and P_Gate Add-On Instructions.


Alarm Name	P_Alarm Name	P_Gate Name	Description
Full Stall	FullStall	None	Raised when the valve has and is using Open and/or Closed limit switches, an attempt is made to open or close the valve, and the limit switches indicate that the valve did not move from its original position at all within the configured time. If Full Stall is configured as a shed fault, the valve is de-energized and a reset is required to command the valve to its energized position.
Interlock Trip	IntlkTrip	None	Raised when the valve is energized and an interlock 'not OK' condition causes the valve to be de-energized. If interlocks are not bypassed, a bypassable interlock or a non-bypassable interlock 'not OK' condition initiates an interlock trip. If interlocks are bypassed, only a non-bypassable interlock 'not OK' condition initiates an interlock trip.
I/O Fault	IOFault	None	Raised when the Inp_IOFault input is true. This input is used to indicate to the instruction that a communication failure has occurred for its I/O. If the I/O Fault is configured as a shed fault, the device transitions to the Faulted state and remains de-energized until reset.
Transit Stall	TransitStall	None	Raised when the valve has and is using both open and closed position feedback, an attempt is made to open or close the valve, and the position feedback indicates that the valve moved off the original position but did not reach the target position within the configured transit stall time.

Parameters of the P_Alarm object can be accessed by using the following convention: [P_Alarm Name].[P_Alarm Parameter].

The Full Stall and Transit Stall Alarms have a configurable delay to allow the open and/or closed feedback time to align with the commanded output. This delay also provides time for the Solenoid-operated valve to open or close.

Simulation

Simulation in P_ValveSO de-energizes the outputs and provides feedback of a working valve. You can manipulate the instruction as if a working solenoid valve were present.

You must set the Inp_Sim parameter in the Controller to '1' to enable simulation. The Simulation or Loopback Test icon  is displayed at the top left of the Operator faceplate, indicating the device is in simulation.

The delay between a command to Open or Close and the simulated opened or closed response is configurable (Cfg_SimFdbkT). (This same delay is used if the Solenoid Valve is configured with no Open/Close feedback.)

When you have finished in simulation, set the Inp_Sim parameter in the controller to '0' to return to normal operation.

Execution

The following table explains the handling of instruction execution conditions.

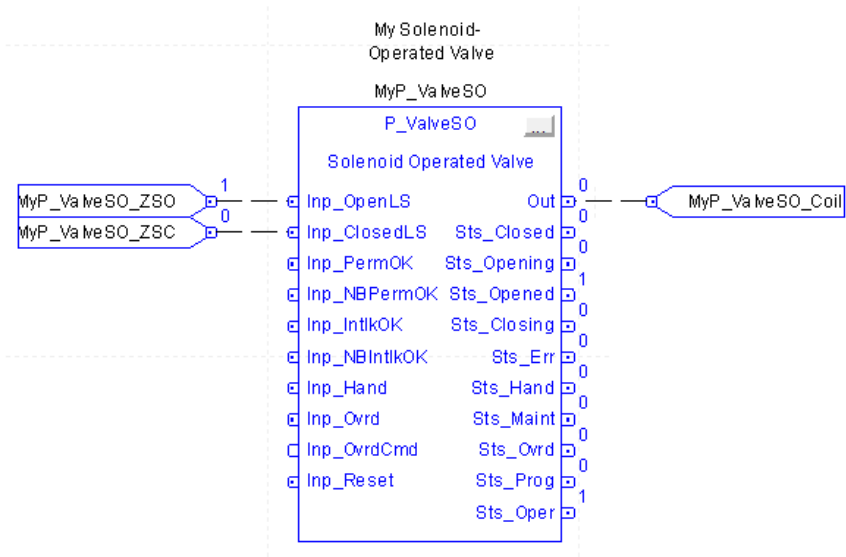
Condition	Description
EnableIn False (false rung)	Processing for EnableIn False (false rung) is handled the same as if the Solenoid Valve were taken out of service by Command. The Solenoid Valve outputs are de-energized and the Solenoid Valve is shown as Program Out of Service on the HMI. All alarms are cleared.
Powerup (prescan, first scan)	The embedded P_CmdSrc and P_Alarm Add-On Instructions handle the processing of Modes and Alarms on prescan and powerup - refer to their specifications for details. On Powerup, the Solenoid Valve is treated as if it had been commanded to its de-energized position.
Postscan (SFC transition)	No SFC Postscan logic is provided.

For more information, see the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#).

Programming Example

The following is a simple example of P_ValveSO.

Boolean parameters MyP_ValveSO_ZSO and MyP_ValveSO_ZSC are used as limit switch feedback inputs and MyP_ValveSO_Coil is used as an output to energize the solenoid.



2-state Valve Statistics (P_ValveStats)

This section is for the operation of the Add-on Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_ValveStats (2-state Valve Statistics) Add-On Instruction monitors a 2-state (open and close) valve and records various statistics related to stroke times and stroke counts. The global object and faceplate shown below are examples of the graphical interface tools for this Add-On Instruction.

Functional Description

The P_ValveStats instruction monitors a 2-state valve and provides the following statistics:

- Amount of time in the current state (closed, opening, opened, closing, stopped/other)
- Amount of time the valve was in each state the last time it was in that state (closed, opening, opened, closing, stopped/other)
- Maximum amount of time that is spent in each state (closed, opening, opened, closing, and stopped/other).
- Total amount of time that is spent in each state (closed, opening, opened, closing, stopped/other)
- Moving average of the last 10 closing (close stroke) times
- Moving average of the last 10 opening (open stroke) times
- Number of completed open strokes (from closed to open)
- Number of completed close strokes (from opened to closed)
- Number of incomplete open strokes (from closed to opening and back to closed)
- Number of incomplete close strokes (from opened to closing and back to opened)
- Number of times the valve was in the stopped/other state
- Number of 'slow' open strokes, the number of open strokes that took longer than the configured Slow Open Time
- Number of 'slow' close strokes, the number of close strokes that took longer than the configured Slow Close Time

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is

defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_ValveStats_4.10.00_AOIL5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Operations

This section describes the primary operations for this Add-On Instruction.

Alarms

The P_ValveStats instruction does not provide any alarms and does not have any embedded P_Alarm Add-On Instructions. Slow Open and Slow Close Status bits are provided if you want to alarm on every slow open stroke or slow close stroke. External P_Alarm instances can be tied to these status outputs.

Simulation

The 2-state Valve Statistics Add-On Instruction does not have a Simulation capability.

It monitors the associated valve regardless of whether that valve is live or simulated.

Execution

The following table explains the handling of instruction execution conditions.

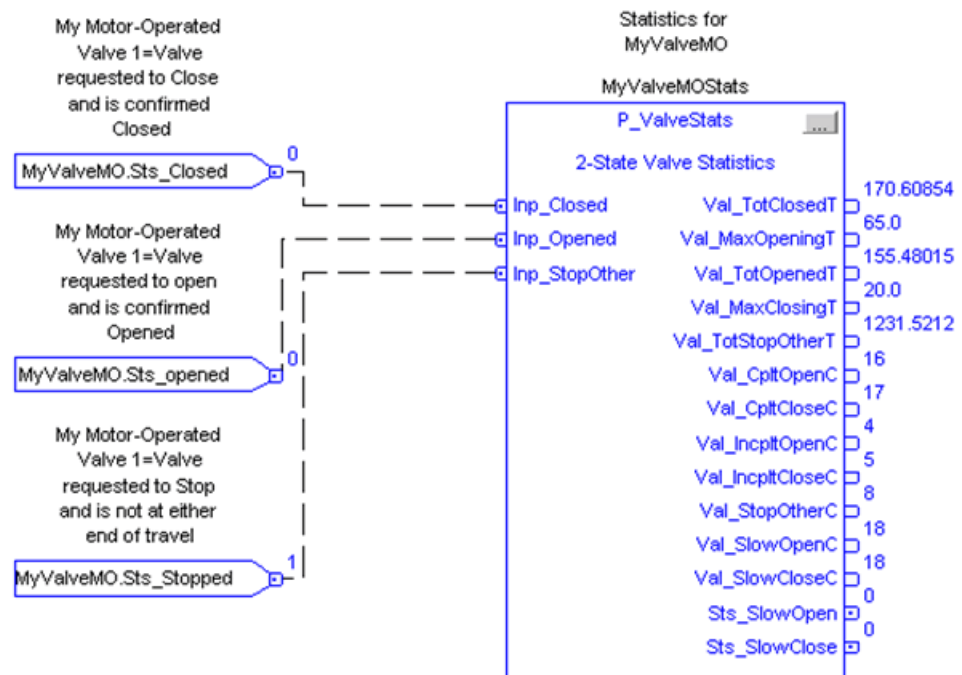
Condition	Description
EnableIn False (false rung)	Total times, total valve stroke counts, and slow stroke counts are maintained. Current position times are cleared. The internal instruction state for the valve is set to 'unknown'.
Powerup (prescan, first scan)	Total times, total valve stroke counts, and slow stroke counts are maintained. Current position times are cleared. The internal instruction state for the valve is set to 'unknown'. Any commands that are received while the controller was in Program command source are cleared.
Postscan	No SFC Postscan logic is provided.

For more information, see the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#).

Programming Example

This section shows how the P_ValveStats instruction can collect statistics on a motor-operated valve. In this example, the motor-operated valve is controlled by using the P_ValveMO instruction. By naming the P_ValveStats instance tag the same as the P_ValveMO tag plus 'ValveStats', it is automatically linked at the HMI to the valve instance.

In this example, the motor-operated valve is either opened, closed, or the motor could stop moving while in travel before reaching either position. Statistics for all of these three states can be tracked by using the P_ValveStats instruction.



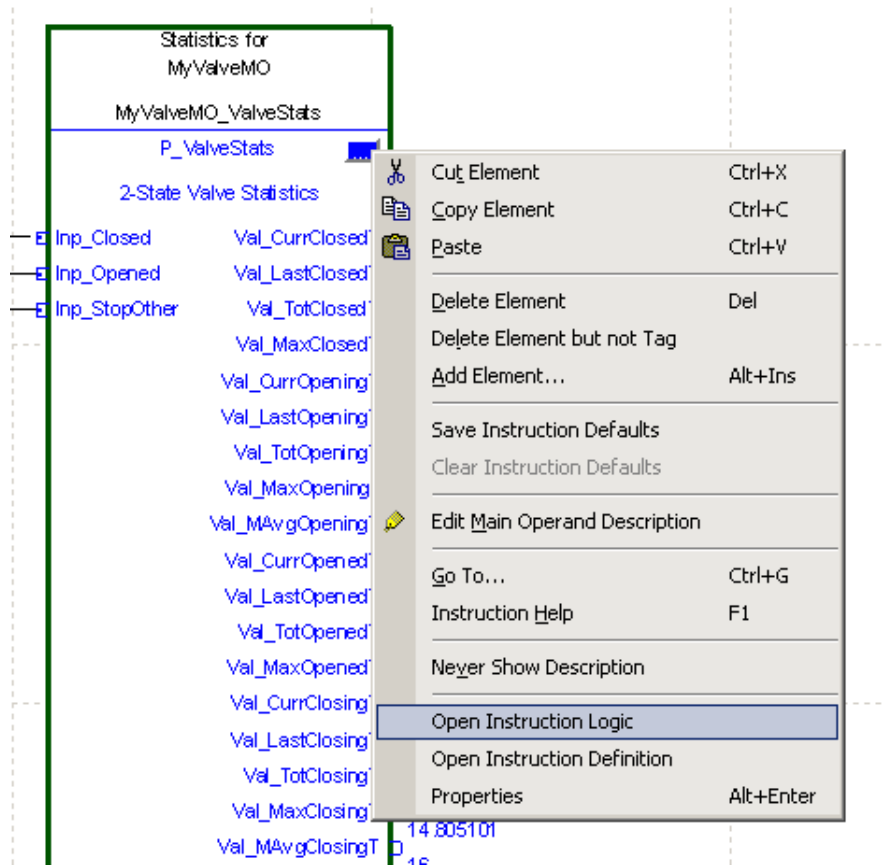
In this example, the parameters Inp_Closed, Inp_Opened, and Inp_StopOther are connected to the parameters Sts_Closed, Sts_Opened, and Sts_Stopped of the P_ValveMO instruction.

The P_ValveStats instruction keeps track of completed strokes, plus open and close strokes that are slower than expected. The parameters Cfg_SlowOpenT and Cfg_SlowCloseT are set to 10, to indicate that any transition longer than 10 seconds is considered slow.

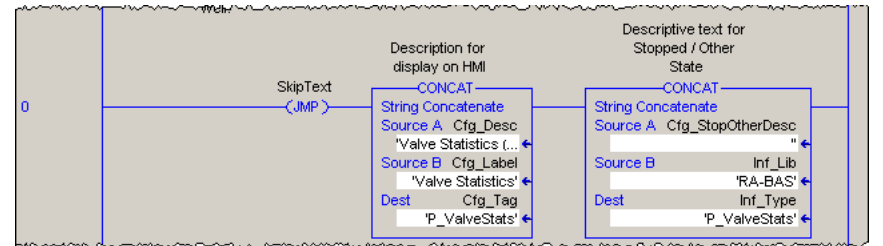
The following local configuration tags are configured to drive the text on the HMI faceplate:

Cfg_Tag:	'MyValveMO_Stats'
Cfg_Label:	'MyValveMO Stats'
Cfg_Desc:	'MyValveMO Statistics'
Cfg_StopOtherDesc:	'Stopped'

Local tags can be configured through the HMI faceplates or in the Logix Designer application by opening the Instruction Logic of the Add-On Instruction instance and then selecting the string on the displayed rung.



All of the strings in local tags are shown on the first rung of the Add-On Instruction's 'Logic' routine for your convenience.



To access the valve statistics from the faceplate for the valve, you must configure MyValveMO. Set the Cfg_HasStatsObj parameter to 1. There is no need to set a global object parameter, but the P_ValveStats backing tag must be named the same as the Valve tag plus '_ValveStats'.

n-Position Device (P_nPos)

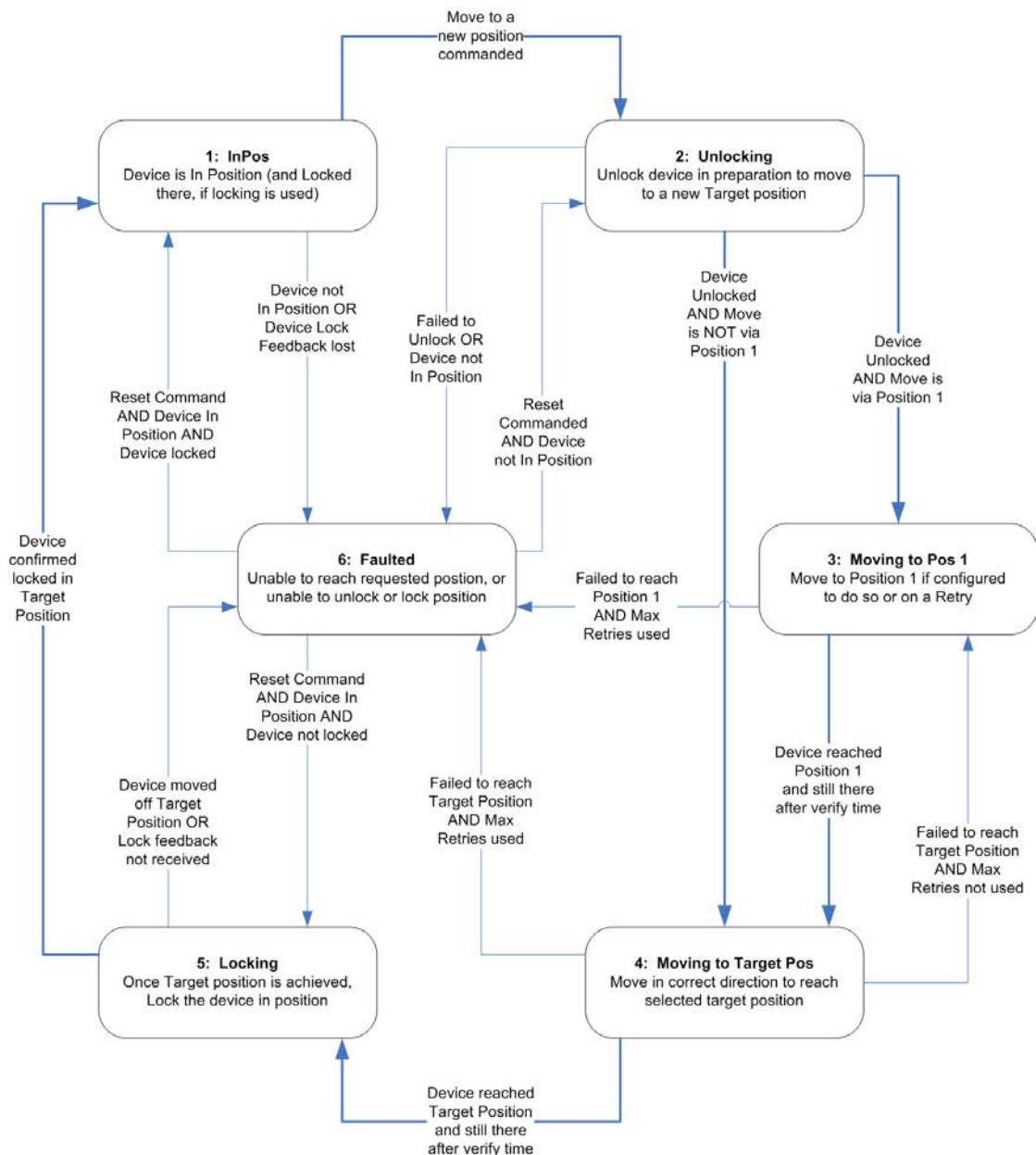
This section is for the operation of the Add-on Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_nPos (n-Position Device) Add-On Instruction controls a circular or linear discrete device with 2...8 positions. The P_nPos instruction provides outputs to select an individual position and outputs to move toward increasing positions ('clockwise' for a circular device) or decreasing positions ('counterclockwise' for a circular device).

For linear devices, the P_nPos instruction can be configured to return to Position 1 on every move, approaching the target position from the 'same side' on each move to improve position repeatability, or move directly to the new position.

For circular devices, the P_nPos instruction can be configured to move only 'clockwise' to increase positions (for example, 6, 7, 8, 1, 2...) or both directions by using the shortest move (for example, 'clockwise' from 6...1: 6, 7, 8, 1; or 'counterclockwise' from 2...7: 2, 1, 8, 7).

The diagram shows the functional characteristics of the P_nPos Add-On Instruction.



Functional Description

The n-Position Device instruction provides the following capabilities:

- Controls and monitors a multi-position device (up to eight positions), such as rotary valves, and other devices with multiple fixed positions
- Monitors limit switches or other position feedback and displays actual device position
- Checks for failure to reach the requested position within a configured time. Provides Alarm on Position Failure

- Monitors Permissive conditions to allow moving to a new position
- Monitors Interlock conditions to de-energize the device, or to request the device to return to Position 1. Provides an Interlock Trip Alarm |if an interlock condition causes the device to de-energize or return to Position 1
- Provides outputs to request each position, and provides outputs for increasing and decreasing position
- Provides outputs to sequence indexing cylinders for devices that use pneumatic or hydraulic devices to step through positions. The cylinders work in an Extend, Shift, Retract, Shift sequence to engage the device, and step it to the next position. The cylinder sequence reverses the Shift directions when driving circular devices 'counterclockwise' (for devices that support bidirectional operation)
- Optionally provides handling of a position lock or seal that must be driven to an unlocked or unsealed state before moving the device, and returned to a locked or sealed state after the move is completed
- Capability for maintenance personnel to take the drive out of service.

IMPORTANT This capability is not a substitute for hard lockout/tagout (LOTO) procedures.

- If the optional lock or seal is used, provides position feedback for the lock or seal to verify the locked or unlocked state at appropriate times. Provides Alarm for Lock Failure
- Provides a simulation capability, responding as if a working device were present while keeping outputs de-energized. The simulation capability can be used for activities such as system testing, operator training, or as part of a full process simulation
- Monitors for I/O communication faults and provides an I/O Fault Alarm
- Provides an 'Available' status for use by automation logic so the logic knows when it has control of the device
- Provides maintenance capabilities, such as the ability to bypass any bypassable interlocks or permissives or temporarily disable feedback checking

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_nPos_4.10.00.AOIL5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Operations

This section describes the primary operations for this Add-On Instruction.

IMPORTANT See [Appendix A](#) for Command Sources and Simulation types.

Alarms


This instruction uses the following alarms, which are implemented by using embedded P_Alarm and P_Gate Add-On Instructions.

Alarm Name	P_Alarm Name	P_Gate Name	Description
Interlock Trip	IntlkTrip	None	Raised when an interlock 'not OK' condition occurs and the device is not in Position 1. The device can be configured to be commanded to Position 1 when an interlock trip occurs. If interlocks are not bypassed, a bypassable interlock or a non-bypassable interlock 'not OK' condition initiates an interlock trip. If interlocks are bypassed, only a non-bypassable interlock 'not OK' condition initiates an interlock trip.
I/O Fault	IOFault	None	Raised when the Inp_IOFault input is true. This input is used to indicate to the instruction that a communication failure has occurred for its I/O. If the I/O Fault is configured as a shed fault, the device transitions to the Faulted state and remains de-energized until reset.
Lock Fail	LockFail	None	Raised when a device with a locking or sealing feature is commanded to a new position, but the lock/seal feedback failed to confirm the device unlocking before moving or failed to confirm the device locking after moving, within the time allowed. If the Lock Failure is configured as a shed fault, the device transitions to the Faulted state and remains de-energized until reset.
Position Fail	PosFail	None	Raised when the device is commanded to a new position, but the device feedback does not confirm that the device reached that position within the configured failure time (Cfg_PosChkT). If the Failure is configured as a shed fault, the device is commanded Off and cannot be commanded On until reset.

Parameters of the P_Alarm object can be accessed by using the following convention: [P_Alarm Name].[P_Alarm Parameter].

Simulation

Simulation in P_nPos disables the normal outputs and provides feedback of a working device. This lets you operate the n_Position Add-On Instruction as if it were a working device, even if no device is physically present.

You must set the Inp_Sim parameter in the Controller to '1' to enable simulation. The Simulation or Loopback Test icon  is displayed at the top left of the Operator faceplate, indicating the device is in simulation.

You can also set the following parameters in simulation:

- Cfg_PosSimT - time to reach target position in simulation (seconds)
- Cfg_LockSimT - time to lock or unlock in simulation (seconds)
- Cfg_CylSimT - time to simulate index cylinder feedback in simulation (seconds)

When you have finished in simulation, set the Inp_Sim parameter in the controller to '0' to return to normal operation.

Execution

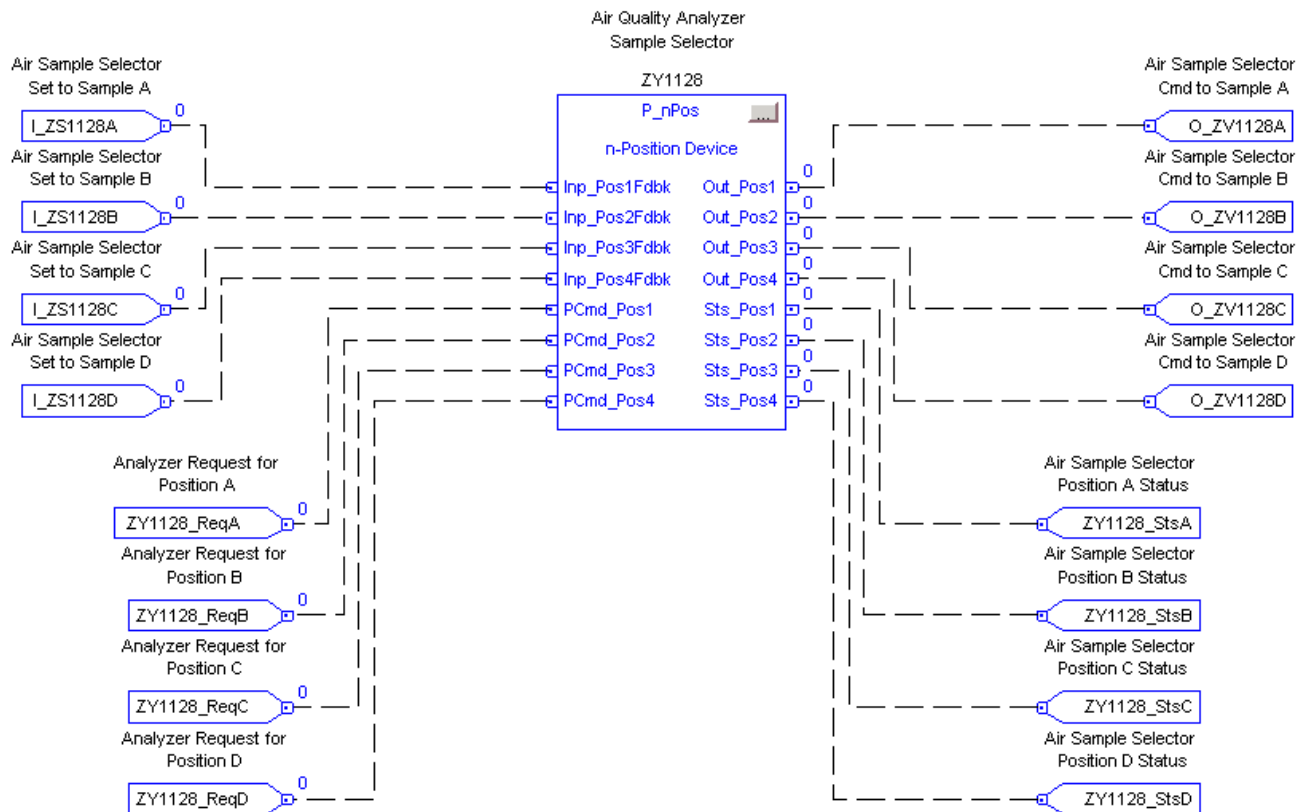
The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	Processing for EnableIn False (false rung) is handled the same as if the device were taken out of service by Command. The device outputs are de-energized and the device is shown as Program Out of Service on the HMI. All alarms are cleared.
Powerup (prescan, first scan)	On prescan, any commands that are received before first scan are discarded. The device is de-energized. On first scan, the device is treated as if it were returning from Hand command source: the instruction state is set based on the position feedback that is received from the device. If the feedback is valid for one position, the device is set to that position, and, if the device has the lock/seal capability enabled, the device is locked in that position. If the device does not have position feedback or the position feedback is invalid, the device is set to the 'unknown/powerup' state. Embedded P_CmdSrc and P_Alarm Instructions are handled in accordance with their standard power-up procedures.
Postscan	No SFC Postscan logic is provided.

For more information, see the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#).

Programming Example

This example uses the P_nPos instruction to control a rotating selector valve with four fixed positions. Each position directs a sample air from one of four sampling locations to an air quality monitor. The rotating selector valve directs all non-selected streams to flow to a common outlet to vent. In this example, the device handles transitions from one position to another. The instruction does not have to enforce a progression of positions to get to the desired state.



First, the instruction is configured to recognize the inputs coming from the selector valve. For this example, the parameter `Cfg_NumPos` is set to 4, indicating this is a four-position device. The parameter `Cfg_HasPosFdbk` and `Cfg_UsePosFdbk` are both set to 1 to indicate that the selector valve provides position feedback, and must be used. The input parameters for positions 1...4 (`Inp_Pos1Fdbk`, `Inp_Pos2Fdbk`, `Inp_Pos3Fdbk`, and `Inp_Pos4Fdbk`) are connected to the digital inputs representing the status of the selector valve.

Next, the instruction is configured to connect to the outputs of the instruction to the selector valve. The parameter `Cfg_OutPosLatch` is set to 1 to latch the output parameter until a new position is commanded. The output parameters for positions 1...4 (`Out_Pos1`, `Out_Pos2`, `Out_Pos3`, and `Out_Pos4`) are connected to the digital outputs that command the selector valve to the desired position.

Once the I/O has been configured, the instruction can be configured to recognize commands from the analyzer control sequence. In this example, the program command parameters for position (PCmd_Pos1, PCmd_Pos2, PCmd_Pos3, and PCmd_Pos4) are connected to the commands from the analyzer control sequence to command the selector valve to the desired position in the sequence.

The instruction is normally operated by sequence program logic. The P_CmdSrc (command source) instruction within the P_ValveC instance must also be configured. The parameter CmdSrc.Cfg_ProgPwrUp is set to 1 so the instruction will power up with the Program command source selected. The parameter CmdSrc.Cfg_ProgNormal is set to 1 to indicate that the instruction will normally use the Program command source. If another command source is selected, the graphic symbol and faceplate will flag the selected source.

The valve does not have a locking or sealing device, so Cfg_HasLock is set to 0.

The parameter Cfg_HasPosFailAlm is set to 1 to indicate that an alarm is desired if the device is not at targeted position (additional settings for the alarm, such as severity and delay timers, are not covered in this example, but must be reviewed and set according to the plant alarm philosophy). Cfg_HasLockFailAlm, Cfg_HasIntlkTripAlm, and Cfg_HasIOFaultAlm are all set to 0, indicating that these alarms are not necessary for this device. The parameter Cfg_PosChkT is set to 30 seconds, to allow 30 seconds for the selector valve to achieve commanded position before a position failure alarm is issued.

The status output parameters (Sts_Pos1, Sts_Pos2, Sts_Pos3, and Sts_Pos4) can be connected to external tags to be used by the analyzer control sequence, if desired.

Lastly, the following local configuration tags must be configured to drive the text on the operations faceplate. In this example, the selector valve P&ID tag is ZY1128. In this example, the strings are set as follows:

Cfg_Tag:	ZY1128
Cfg_Label:	Air Sample Selector
Cfg_Desc:	Air Quality Analyzer Sample Selector
Cfg_Pos1Name:	Position A
Cfg_Pos2Name:	Position B
Cfg_Pos3Name:	Position C
Cfg_Pos4Name:	Position D

Cross Functional

Purpose

This chapter is for the operation of the Add-on Instructions. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The process objects in this group are often used to extend the functionality of other objects. However, they can also be used as standalone objects when necessary to implement a desired control scheme.

[Table 19](#) describes the objects in this group, including when to use and not to use each one.

Table 19 - Cross Functional Objects

Process Object	Description	When to Use	When Not to Use
Condition Gate Delay (P_Gate)	<p>This instruction provides a 'gate' for a discrete signal and provides on-delay and off-delay timing for the gated signal. P_Gate is used within P_DIn, all analog inputs, and P_PIDE for threshold and target disagree status processing.</p> <p>When the gate input is true, the input is passed through to the output with on-delay and off-delay timing applied.</p> <p>When the gate input is false, the output is kept off (the off-delay still applies).</p>	<ul style="list-style-type: none"> This instruction is integrated with the following Rockwell Automation® Library Objects: <ul style="list-style-type: none"> Basic Analog Input (P_AIn) Advanced Analog Input (P_AInAdv) Dual Sensor Analog Input (P_AInDual) Multiple Analog Input (P_AInMulti) Discrete Input Object (P_DIn) Enhanced PID (P_PIDE) 	—
Interlocks with First Out and Bypass (P_Intlk)	<p>This instruction collects the interlock conditions that stop or de-energize a running or energized piece of equipment. The instruction helps prevent the equipment from starting or being energized.</p> <p>Interlocks are always evaluated to de-energize equipment. Configurable text descriptions of shutdown conditions are available.</p> <p>For permissive conditions that must be made to start the equipment, but are ignored once the equipment is running, use the Permissives (P_Perm) instruction.</p>	<ul style="list-style-type: none"> Have multiple interlock conditions or cascaded interlock conditions (an interlock hierarchy) that stop some equipment (motor, valve, drive) or keep it from starting. Link the conditions to P_Intlk inputs, and link the P_Intlk status bits to the Inp_IntlkOK and Inp_NBIntlkOK inputs of the equipment. Need a first-out indication of the interlock condition that shuts down the equipment. Need configurable text descriptions of shutdown conditions or other features of the P_Intlk faceplate. 	<ul style="list-style-type: none"> Have conditions that help prevent the equipment from starting, but are ignored once the equipment is running. These conditions are permissive, not interlock conditions. Use the P_Perm instruction. Have one interlock condition only for the equipment. Connect the condition directly to the interlock input on the device.

Table 19 - Cross Functional Objects

Process Object	Description	When to Use	When Not to Use
Interlocks with First Out and Bypass - Advanced (P_IntlkAdv)	This instruction collects the interlock conditions that stop or de-energize a running or energized piece of equipment. The instruction helps prevent the equipment from starting or being energized. Interlocks are always evaluated to de-energize equipment. Configurable text descriptions of shutdown conditions are available. The instruction helps categorizing interlocks. Each category description are configurable and each category will have a set configuration for bypass and must reset. For permissive conditions that must be made to start the equipment, but are ignored once the equipment is running, use the Permissives (P_Perm) instruction.	<ul style="list-style-type: none"> • Have multiple interlock conditions or cascaded interlock conditions (an interlock hierarchy) that stop some equipment (motor, valve, drive) or keep it from starting. Link the conditions to P_Intlk inputs, and link the P_Intlk status bits to the Inp_IntlkOK, Inp_NBIntlkOK and Inp_IntlkAvail inputs of the equipment. • Need a first-out indication of the interlock condition that shuts down the equipment. • Need configurable text descriptions of shutdown conditions or other features of the P_Intlk faceplate. • Need categorized interlocks • Need to affect the process control availability, based on interlock type • Need to IO Fault interlock condition per interlock input 	<ul style="list-style-type: none"> • Have conditions that help prevent the equipment from starting, but are ignored once the equipment is running. These conditions are permissive, not interlock conditions. Use the P_Perm instruction. • Have one interlock condition only for the equipment. Connect the condition directly to the interlock input on the device. • Don't need categorized interlocks, affecting the process control availability, based on interlock type or IO Fault interlock condition per interlock input
Permissives with Bypass (P_Perm)	This instruction collects the permissive conditions that allow a piece of equipment to start (run, energize, open, and so forth). Permissive conditions generally must be true to start the equipment. Once the equipment is running, permissives are ignored. Use the Interlocks (P_Intlk) instruction to collect conditions that stop running equipment and help prevent it from starting.	<ul style="list-style-type: none"> • Have multiple or cascaded permissive conditions that help prevent some equipment (motor, valve, drive) from starting, but are ignored once the equipment is running. Link the P_Perm instruction status bits to the Inp_PermOK and Inp_NBPermOK inputs of the motor, valve, or drive. • Want configurable text descriptions of the permissive conditions and other features of the P_Perm instruction faceplates. 	<ul style="list-style-type: none"> • Have conditions that shut down running equipment and help prevent it from starting. These conditions are interlock conditions, not permissive conditions. Use the P_Intlk instruction. • Have one permissive condition only for the equipment. You can connect the condition directly to the permissive input on the device.

Table 19 - Cross Functional Objects

Process Object	Description	When to Use	When Not to Use
Discrete 2-, 3-, or 4-state Device (P_D4SD)	<p>This instruction controls and monitors feedback from a discrete 2-state, 3-state, or 4-state device s, monitoring for fault conditions. These devices include multiple-speed motors or multiple-position valves.</p> <p>Controls four discrete outputs and monitors four discrete feedback inputs. Each output and input has configurable states of each output in the various device states.</p> <p>The instruction also monitors permissive and interlock conditions; the latter returns the device to its default state.</p>	<ul style="list-style-type: none"> Need to operate a discrete device that has two, three, or four unique states, and the device is not supported by other Rockwell Automation Library of Process Objects Add-On Instructions for various motors, valves, and so on. Have a device, such as a valve or motor, that is supported by other Add-On Instructions, but you want the device to use non-standard state names. For example, 'recycle' and 'deliver' for a diverter valve rather than the fixed names used in the other instruction, such as 'closed' and 'open'. The P_D4SD Instruction has configurable names for each of the device states. 	<ul style="list-style-type: none"> Operating a device that has more than four states, such as a six-position rotary selector valve. You can use the P_nPos (n-Position Device) Add-On Instruction instead. Operating a single-speed motor, two-speed motor, simple reversing motor, solenoid valve, motor-operated valve, mix-proof valve, or other device that is better supported by other Rockwell Automation Library of Process Objects Add-On Instructions. Instructions such as P_Motor, P_MotorRev, P_Motor2Spd, P_ValveSO, P_ValveMO, and P_ValveMP more closely model the device under control and can provide better diagnostics for the device. Operating a continuously variable device. Use the P_AOut (Analog Output), P_ValveC (Control Valve), or P_VSD (Variable Speed Drive) Add-On Instruction instead. Operating a two-state device that requires pulsing (single-pulse or continuous). Use the P_DOut (Discrete Output) Instruction instead.
Central Reset (P_Reset)	<p>This instruction provides a central point for resetting equipment faults. Latched alarms can be reset for a control strategy.</p> <p>The P_Reset instruction accepts an Operator Reset command, a Program Reset command, and a Reset Input that can come from a push button input. You can also accept a reset output from a higher-level P_Reset instruction.</p> <p>The P_Reset instruction includes a Reset Required input for collecting the Ready to Reset outputs of the various instructions it resets. The instruction provides a Ready to Reset (reset required) status that can illuminate a push button or make an HMI Reset button visible.</p> <p>The P_Reset instruction includes a timer function that causes its output to be held on for at least a minimum time. This functionality lets the reset signal be sent via physical output cards to field devices that require it (for example, motor drives) and gives time for the cleared status from the device to propagate back to Interlock or Permissive inputs.</p>	<ul style="list-style-type: none"> Want a common reset point (Master Reset) for alarms and fault conditions for a control strategy, process unit, process cell or equipment group, process area or plant section, or even a small site. Tie the output of the P_Reset instruction to the Inp_Reset input of the equipment to be reset. Want a cascading reset strategy, where there is a P_Reset instruction for a small equipment scope (such as a strategy) that incorporates resets from wider scope (unit, cell, area, site) resets. Tie the output of the higher-level P_Reset instruction to the Inp_Reset input of the lower-level P_Reset instruction. 	<ul style="list-style-type: none"> Want to reset one piece of equipment (valve, motor). Use the Operator or Program Reset command directly on the equipment.

Table 19 - Cross Functional Objects

Process Object	Description	When to Use	When Not to Use
Common Alarm Block (P_Alarm)	This instruction is used to provide notification to operators of abnormal conditions or events. The instruction handles alarm acknowledgement, alarm reset, alarm shelving/disabling, and alarm suppression (for the FactoryTalk® Alarms and Events server).	<ul style="list-style-type: none"> Developing your own Add-On Instruction and you want it to generate one or more alarms that are compatible with the alarm strategy for the Process Add-On Instructions. Use an instance of the P_Alarm instruction embedded within your Add-On Instruction for each alarm condition. Have a condition in your logic (outside of any Add-On Instruction) that you want to generate an alarm. Use the P_Alarm instruction standalone within your program logic. 	—
Command Source (P_CmdSrc)	This instruction provides selection of the source of commands and settings for an instruction or control strategy. Many other library instructions already incorporate the P_CmdSrc instruction.	<ul style="list-style-type: none"> Use for a device that requires separate acquisition by an operator and program logic, or that supports External, Override, or Hand capabilities, or that needs a separate Maintenance command source. Embed the P_CmdSrc instruction within your Add-On Instruction. Creating a Control Strategy and want standard arbitration between Operator and Program command sources (and perhaps External, Override, Maintenance, or Hand). Use the P_CmdSrc instruction standalone within your strategy, and condition the commands and actions in your strategy on the command source status bits from the P_CmdSrc instruction. 	<ul style="list-style-type: none"> Creating an Add-On Instruction that does not do anything differently for operators versus program logic, for External, Override, or Hand conditions, or for maintenance personnel. You do not need command sources or the P_CmdSrc instruction. Creating a complex strategy for shared equipment (shared use common resource) that has complex rules for arbitration and allocation of the equipment. You need rule-based sharing logic beyond the capabilities of the P_CmdSrc instruction.
Operator Prompt (P_Prompt)	This instruction is a universal mechanism for operator interaction that can be used within a control scheme. The instruction prompts an operator with configurable message or data fields and accepts operator response data and confirmation.	<ul style="list-style-type: none"> Use a prompt to request input from an operator. The input can be any of the following: <ul style="list-style-type: none"> Acknowledging the prompt Viewing and confirming data Making a selection Entering numeric data Entering text data 	<ul style="list-style-type: none"> An alarm, per ANSI/ISA-18.2-2016, is used to notify an operator of an abnormal situation that requires a response. An alert is used to notify an operator of an abnormal situation that does not require a response. (A prompt requires a response, but does not advise of an abnormal situation.)

Table 19 - Cross Functional Objects

Process Object	Description	When to Use	When Not to Use
Boolean Logic with Snapshot (P_Logic)	<p>This instruction executes up to eight gates of configurable Boolean logic. Gate types available include AND, OR, XOR (Exclusive-OR), Set/Reset, Select, and Majority. Each gate provides up to four input conditions that are individually invertible. (The P_Logic instruction does not need a NOT gate.)</p> <p>The instruction also provides a snapshot capability, enabling it to record its current state (with an optional time stamp) upon change in output state, on Operator or Program command, or based on a logic loopback input</p>	<ul style="list-style-type: none"> Want to implement an Interlock or Permissive condition that is more complicated than the simple OR-ing or AND-ing provided by the P_Intlk (Interlocks) or P_Perm (Permissives) Add-On Instructions. Want to implement some Boolean (combination) logic that can be reconfigured from the HMI online, or which requires the snapshot capability for saving a copy of the logic state with a time stamp. Have more than the 16 interlock conditions or permissive conditions provided by the P_Intlk and P_Perm Add-On Instructions, but some of the conditions can be grouped under one identification. For example, the bearing over-temperature signals for a pump and motor (pump inboard bearing, pump outboard bearing, motor inboard bearing, and motor outboard bearing) can be ORed together in a P_Logic instruction. The result is presented to a P_Intlk instruction as one 'Bearing Overtemp' condition. 	<ul style="list-style-type: none"> Need to implement simple interlocks and permissives that are handled by the P_Intlk and P_Perm instructions directly. These instructions can permit operation or trip operation on either the low- or high-state of a condition (configurable inverting). Require logic that is beyond the P_Logic Add-On Instruction capabilities or which is extremely time critical. The P_Logic instruction provides only eight inputs, eight gates, and one output with on-delay and off-delay timing, and it is implemented with table-driven code. Use hard-coded logic in native controller languages instead. The native programming languages are faster and provide functionality beyond what the P_Logic instruction can do.

Condition Gate Delay (P_Gate)

This section is for the operation of the Add-on Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_Gate (Condition Gate Delay) Add-On Instruction provides a 'gate' for a discrete signal and provides on-delay and off-delay timing for the gated signal.

P_Gate is used within P_DIn, all Analog inputs, and P_PIDE for threshold and target disagree status processing.

When the gate input is true, the input is passed through to the output with on-delay and off-delay timing applied. When the gate input is false, the output is kept off (the off-delay still applies).

Functional Description

The primary operations of the P_Gate Add-On Instruction and the faceplates are:

- Provides gate input to turn on and off input monitoring
- Provides Gate delay to define the time the gate input must be on for status outputs to be enabled
- Provides on and off delays for the status output
- Provides configurable strings for gate and status condition descriptions
- Provides faceplate for setup of gate delay, on delay, off delay, and strings

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_Gate_4.10.00_AOIL5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Operations

This section describes the primary operations for this Add-On Instruction.

Alarms

The P_Gate Add-On Instruction does not generate any alarms. In many applications, status bits from P_AIn Analog Input or P_DIn Discrete Input instructions are sent to the P_Gate inputs. The output of P_Gate is the input to a P_Alarm instruction.

Simulation

The P_Gate Add-On Instruction does not have a Simulation capability.

Execution

The following table explains the handling of instruction execution conditions.

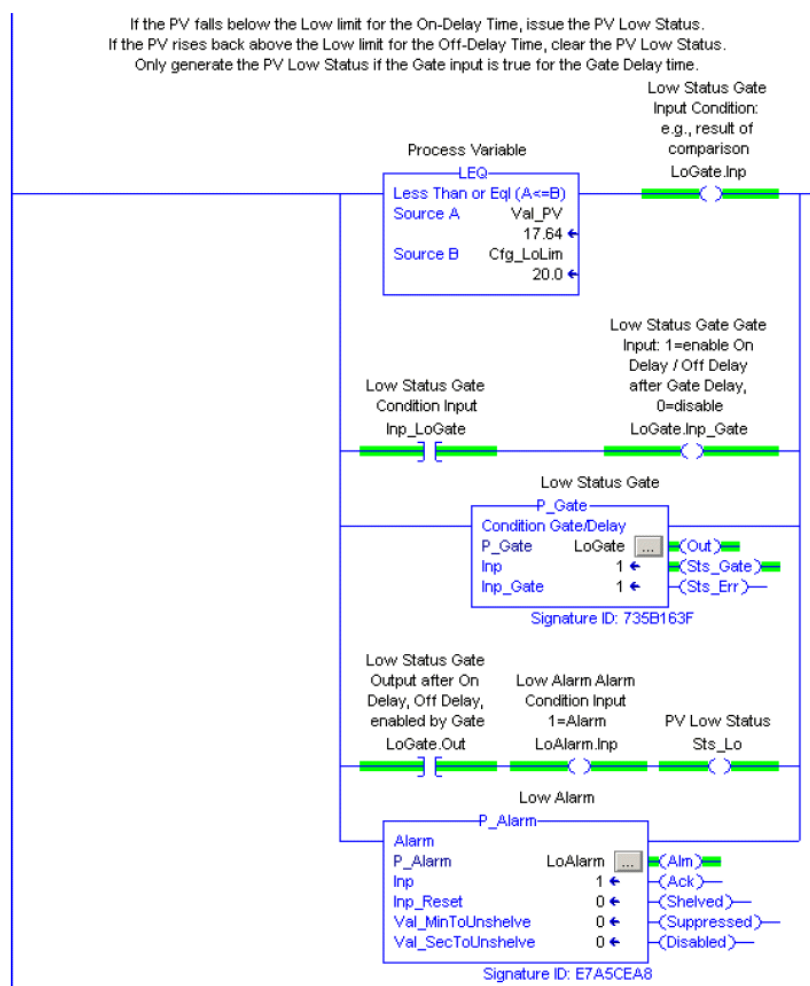
Condition	Description
EnableIn False (false rung)	Processing for EnableIn False is handled the same as the main logic routine except that the state of Inp is inverted. This inversion allows the P_Gate Add-On Instruction in a ladder diagram instance to have its condition input mapped by using the rung condition instead of separate mapping logic. Be sure that the Inp parameter is set to 1 (its default value) when using rung condition mapping. See Implementation by Using the EnableIn False Feature on page 338 for details. IMPORTANT: Only the condition input 'Inp' is inverted when EnableIn is false. The gate input 'Inp_Gate' must still be mapped.
Powerup (prescan, first scan)	During Prescan, the Inp_Gate input is set to 1. This setting makes sure that the condition input Inp is processed with on-delay and off-delay timing if the Inp_Gate pin becomes unconnected or unreferenced.
Postscan (SFC Transition)	No SFC postscan logic is provided.

For more information, see the Logix 5000™ Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#).

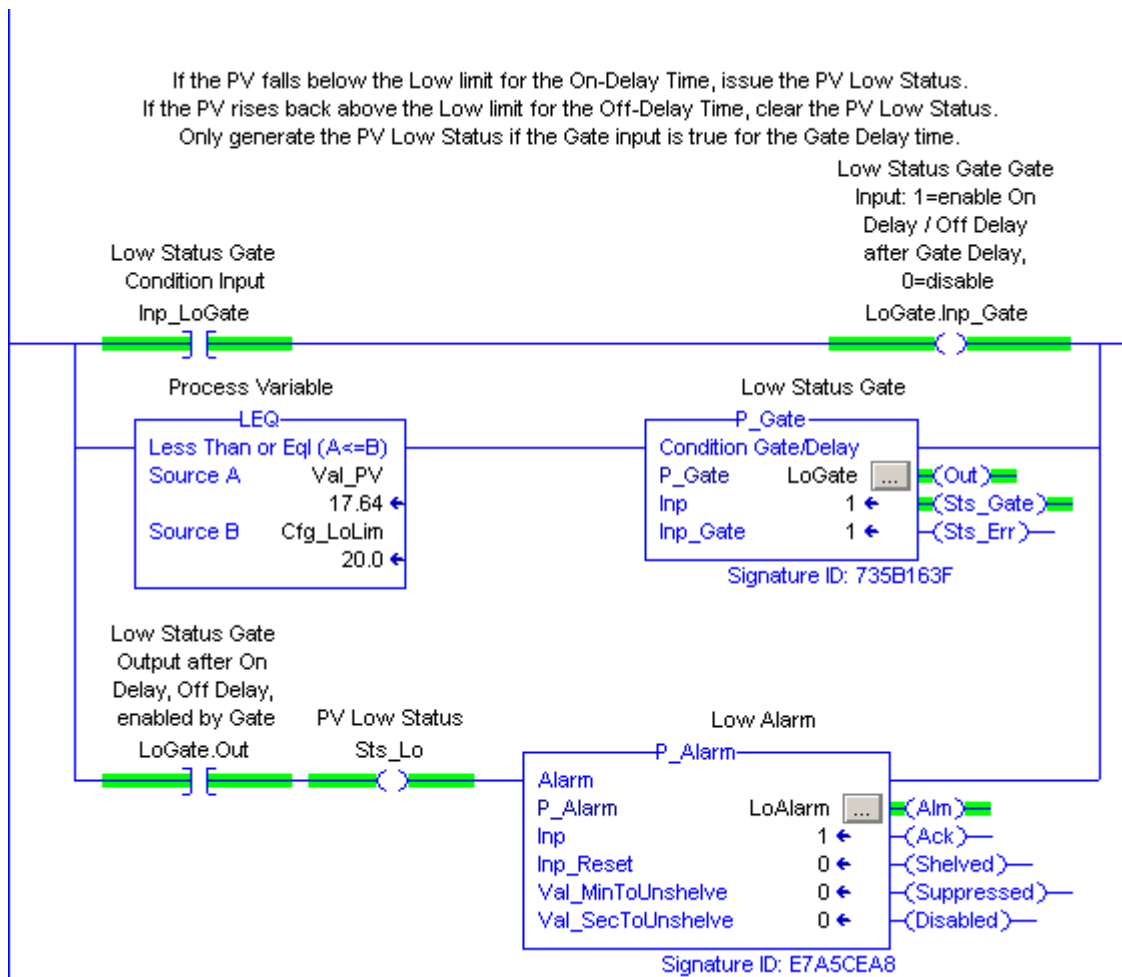
Implementation by Using the EnableIn False Feature

The P_Gate instruction can be used in a Ladder Diagram routine with the input condition carried by the Rung-In state instead of being mapped by separate logic, such as an XIC/OTE branch.

The following illustration shows normal implementation with the condition input and gate input mapped to the P_Gate instance by logic, in this case on the branches immediately before the instance.



The following illustration shows the EnableIn False implementation, with the condition input mapped by using the Rung-In state. (It also shows Rung-In mapping for the P_Alarm instruction.)



Note that whether or not you map the P_Gate instruction's condition input by using the Rung-In state, the Inp_Gate signal must still be mapped by logic. Only the condition input Inp follows the Rung-In state.

The Rung-In condition determines whether the Add-On Instruction's normal code ('Logic' Routine) is executed or its EnableIn False code ('EnableInFalse' Routine). The two routines use identical code, except that in the EnableInFalse Routine all references to Inp are inverted.

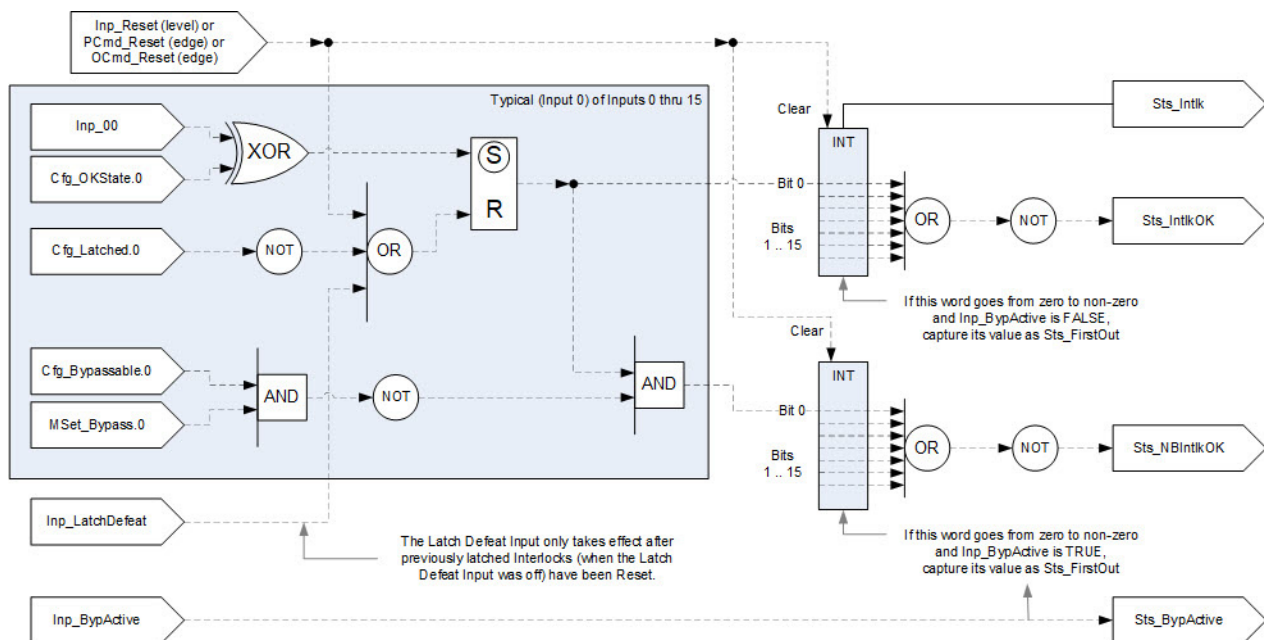
When using the Rung-In mapping method, be sure to set Inp to 1 (its default value). Then, when the rung is TRUE, Inp (which is 1) is treated as TRUE. When the rung is FALSE, Inp (which is still 1) is treated as FALSE (inverted).

Interlocks with First Out and Bypass (P_Intlk)

This section is for the operation of the Add-on Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_Intlk (Interlocks with First Out and Bypass) Add-On Instruction is used to collect (sum up) the interlock conditions that stop or de-energize a running or energized piece of equipment and prevent it from starting or being energized. Interlocks are always evaluated to de-energize equipment. For permissive conditions that must be made to start the equipment, but are ignored once the equipment is running, use the Permissives (P_Perm) Add-On Instruction.

The following diagram shows the functional characteristics of the P_Intlk Add-On Instruction.



Functional Description

The following are the primary operations of this instruction and faceplate:

- **Interlock Input OK Check:** Each input is compared with its configured OK state. If the input is not in its OK state, it raises an interlock condition unless bypassed by Maintenance.
- **Interlock Condition Latching:** If the input is configured as latched, the interlock condition is latched in until reset unless the latch defeat input is true. If the input is not configured as latched, the interlock condition clears when the input is again in its OK state.
- **Interlock Bypass:** If an input is configured as able to be bypassed ($\text{Cfg_Bypassable.x} = 1$), the interlock is bypassed ($\text{MSet_Bypass.x} = 1$), and the downstream equipment instruction has bypass enabled ($\text{Sts_Bypass} = 1$), the input will not raise an interlock condition, even if it is not in its OK state. If the input is configured as not able to be bypassed ($\text{Cfg_Bypassable.x} = 0$) or if interlock is not bypassed ($\text{MSet_Bypass.x} = 0$) or the connected device does not have bypass enabled ($\text{Sts_Bypass} = 0$), the input will raise an interlock condition.

Typically, Engineering configures which interlocks are allowed to be bypassed. Maintenance then picks the ones to bypass from the interlocks that are allowed by Engineering.

- **First Out:** If no interlock conditions are raised (OK to run), the first interlock condition to be raised is marked as the first out. If multiple such interlock conditions are raised in the same scan, they are all marked as first out.
- **Latch Defeat:** A latch defeat function is provided to reduce the number of operator actions that are required to start equipment. The latch defeat input is set when the equipment is not running. When the latch defeat input is true, the latched configuration of inputs is ignored, and all interlock conditions clear when their corresponding inputs are in their OK states. This action saves the operator from having to reset before starting the equipment. When the equipment starts, the latch defeat input is turned off. Then, if an interlock condition configured as latched shuts down the equipment, it remains latched until reset.

IMPORTANT To help prevent loss of information about what caused the equipment to shut down, the latch defeat input is not processed until after any latched interlocks (that occurred when the latch defeat was off) have been reset.

- **Summary Status:** The instruction summarizes its 16-interlock input conditions into two primary status bits: Sts_IntlkOK and Sts_NBIntlkOK. Sts_IntlkOK indicates that all interlock conditions are clear regardless of bypass state (ready to run regardless of bypassed state). Sts_NBIntlkOK indicates that all interlock conditions that cannot be bypassed are clear and all interlock conditions that are able to be bypassed are either clear or bypassed (ready to run if interlocks are bypassed).

IMPORTANT The downstream equipment instructions determine whether interlocks are bypassed. The P_Intlk instruction simply provides the two summary status bits. These 2 bits are to be wired or mapped to the equipment control logic.

- **Faceplate:** The instruction faceplate displays the interlock condition state of each input and whether it is bypassed, and shows the overall interlock (summary) status. The Engineering tab of the faceplate lets you configure the instruction for the following:
 - OK state configuration
 - Latch configuration
 - Configuration of interlocks that can be bypassed, and the text that is associated with each interlock condition input.

You can also configure a navigation tag for each interlock condition. If you enable navigation, the condition text on the Operator tab of the faceplate can be clicked to access the faceplate for the corresponding tag

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This code lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_Intlk_4.10.00_AOIL5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Operations

This section describes the primary operations for this Add-On Instruction.

Alarms

The P_Intlk Add-On Instruction does not generate any alarms. The individual input conditions can be alarmed, if necessary, in other logic before they are sent to the inputs of the P_Intlk instruction. In many applications, status bits from P_AIn Analog Input or P_DIn Discrete Input instructions are sent to the P_Intlk inputs, and those instructions provide alarms.



ATTENTION: Status bits should be used as interlock conditions. Use Alarm bits as interlock conditions only if you intend that the interlock condition be ignored when the corresponding alarm is disabled, suppressed, or shelved.

Simulation

The P_Intlk Add-On Instruction does not have a Simulation capability.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	Processing for EnableIn False (false rung) is handled by setting the summary All interlocks OK and All interlocks (cannot be bypassed) OK status outputs to false (0). The individual interlock bit status and first out outputs are left in their last state.
Powerup (prescan, first scan)	The latch and first out states of the P_Intlk Add-On Instruction are maintained through a power-down/power-up or Run/Program/Run cycle. Any commands that are received before the first scan are discarded.
Postscan (SFC transition)	No SFC postscan logic is provided.

For more information, see the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#).

Programming Example

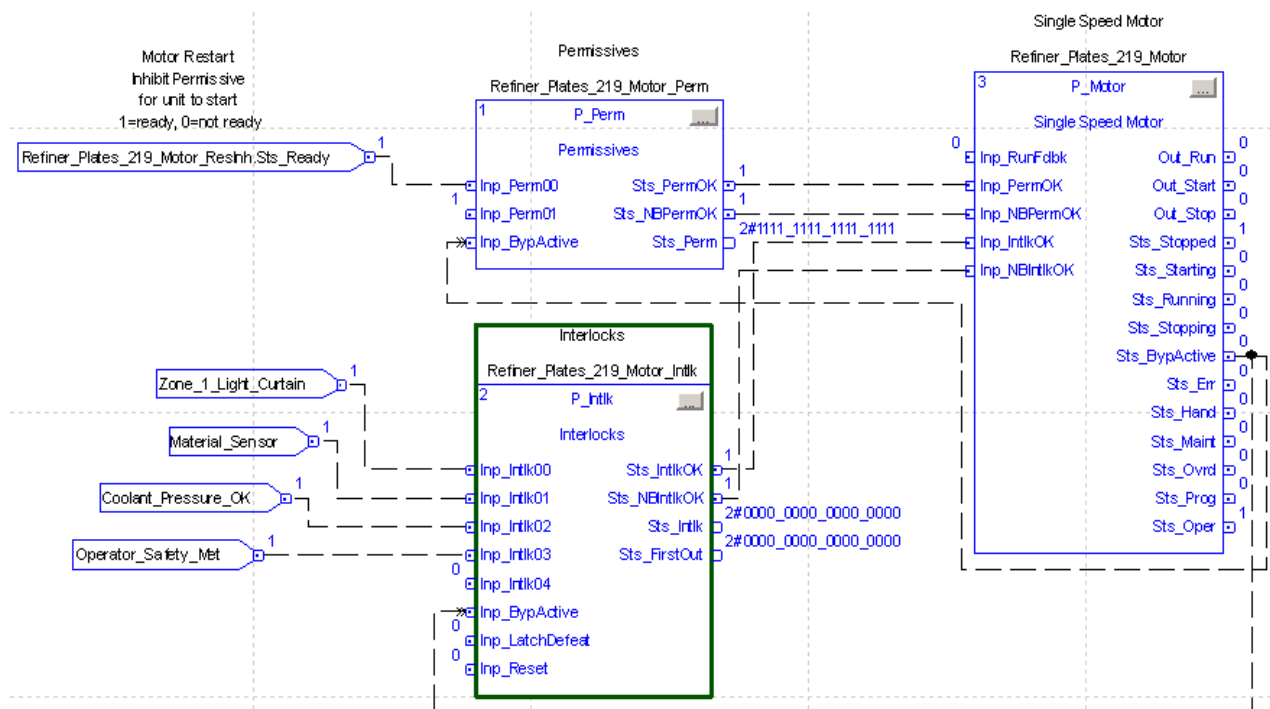
This example uses the P_Intlk instruction to concentrate the interlock conditions that allow the functioning of the refiner plates that are used for grinding wood as part of the pulp manufacturing process.

Perform the following steps to import an Add-On Instruction to your project.

1. Right-click 'Add-On Instructions' and select 'Import Add-On Instruction...'

2. On the Import Add-On Instruction dialog box, select the P_Intlk instruction and click Import.
3. Add the P_Intlk instruction to your project:
 - a. Click the Add-On tab on the Language Element toolbar.
 - b. Click the P_Intlk instruction.
 - c. Also add the P_Motor instruction that controls the refiner plates.
4. Double-click the interlock instruction name and create the tag name for it. The naming convention that makes navigation from the motor faceplate work automatically is to use the motor's tag name followed by _Intlk.
5. Create the input references necessary to ensure the appropriate operation of the refiner plates and create their appropriate tags.
6. Expose the Sts_BypActive pin on P_Motor. Wire this pin back to Inp_BypActive on P_Intlk, and mark this wire 'Assume Data Available'.

The following image is what the instruction looks like when connected correctly.



7. Save, download, and run your Logix application.
8. In your HMI application, add a P_Motor motor graphic object from P_Motor Graphics Library.ggfx and right-click it.
9. From the list, select "Global Object Parameter Values"
10. Associate the created tag for P_Motor of the Refiner Plates in the controller with the graphic object in the HMI:
 - a. On the first line of the Global Object Parameter Value table, type its name.

- b. On the second line, type the name of the shortcut to the controller enclosed by brackets.
- c. Fill in additional parameters as desired.

Because of the naming convention, the connection will be made automatically and the interlock settings will be associated with the refiner plates motor.

- 11. Save and your HMI application.
- 12. Run your HMI application and click the motor graphic object. If the Quick display appears, click the 'go to faceplate' button.

On the P_Motor faceplate, click the HMI configuration tab and go to page 2.

- 13. Check "Enable navigation to interlock object".

The Interlock button is enabled.

- 14. On the P_Motor faceplate, click the Interlock button to open the P_Intlk faceplate. On the P_Intlk faceplate, click the Engineering tab, and perform the following:
 - a. Name the interlocks accordingly.
 - b. Select the appropriate state under 'OK State'.
 - c. Indicate which interlocks can be bypassed.
 - d. Indicate which interlocks must be reset.

TIP Specific inputs can be bypassed on the maintenance tab based on the selections.

When bypass is enabled, it bypasses only those set on the Maintenance tab.

Interlocks with First Out and Bypass - Advanced (P_IntlkAdv)

This section is for the operation of the Add-On Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

Functional Description

The following are the primary operations of this instruction and faceplate:

- **Interlock Input OK Check:** Each input is compared with its configured OK state. If the input is not in its OK state, it raises an interlock condition unless bypassed by Maintenance.
- **Interlock Condition Latching:** If the input is configured as latched, the interlock condition is latched in until reset unless the latch defeat input is true. If the input is not configured as latched, the interlock condition clears when the input is again in its OK state.
- **Interlock Bypass:** If an input is configured as able to be bypassed ($\text{Cfg_Bypassable.x} = 1$), the interlock is bypassed ($\text{MSet_Bypass.x} = 1$), and the downstream equipment instruction has bypass enabled ($\text{Sts_Bypass} = 1$), the input will not raise an interlock condition, even if it is not in its OK state. If the input is configured as not able to be bypassed ($\text{Cfg_Bypassable.x} = 0$) or if interlock is not bypassed ($\text{MSet_Bypass.x} = 0$) or the connected device does not have bypass enabled ($\text{Sts_Bypass} = 0$), the input will raise an interlock condition.

Typically, Engineering configures which interlocks are allowed to be bypassed. Maintenance then picks the ones to bypass from the interlocks that are allowed by Engineering.

- **First Out:** If no interlock conditions are raised (OK to run), the first interlock condition to be raised is marked as the first out. If multiple such interlock conditions are raised in the same scan, they are all marked as first out.
- **Latch Defeat:** A latch defeat function is provided to reduce the number of operator actions that are required to start equipment. The latch defeat input is set when the equipment is not running. When the latch defeat input is true, the latched configuration of inputs is ignored, and all interlock conditions clear when their corresponding inputs are in their OK states. This action saves the operator from having to reset before starting the equipment. When the equipment starts, the latch defeat input is turned off. Then, if an interlock condition configured as latched shuts down the equipment, it remains latched until reset.

IMPORTANT To help prevent loss of information about what caused the equipment to shut down, the latch defeat input is not processed until after any latched interlocks (that occurred when the latch defeat was off) have been reset.

- **Interlock Types:** There are 8 configurable interlock Types. Each Type has a description; by default, the description is a 3-letter abbreviation (for HMI use) followed by a colon and the full description. Each Type affects bypassability, latching and availability. Types 0, 2, 4 and 6 affect availability; Types 0, 1, 2 and 3 force latching and require reset; Types 0, 1, 4 and 5 force not allowing bypass.
- **Input Available Field:** An input Available field allows for cascading of interlock banks and working with equipment that is ready to run but not available at certain points in a sequence (for example, a pump with an automated block valve that is currently closed).
- **Summary Status:** The instruction summarizes its 16-interlock input conditions into three primary status bits: Sts_IntlkOK, Sts_NBIntlkOK, and Sts_Avail. Sts_IntlkOK indicates that all interlock conditions are clear regardless of bypass state (ready to run regardless of bypassed state). Sts_NBIntlkOK indicates that all interlock conditions that cannot be bypassed are clear and all interlock conditions that are able to be bypassed are either clear or bypassed (ready to run if interlocks are bypassed). Sts_Avail indicates to equipment that it is still available to be energized although an interlock is currently active.

IMPORTANT The downstream equipment instructions determine whether interlocks are bypassed. The P_Intlk instruction simply provides the two summary status bits. These 2 bits are to be wired or mapped to the equipment control logic.

- **Faceplate:** The instruction faceplate displays the interlock condition state of each input and whether it is bypassed, and shows the overall interlock (summary) status. The Engineering tab of the faceplate lets you configure the instruction for the following:
 - OK state configuration
 - Latch configuration
 - Configuration of interlocks that can be bypassed, and the text that is associated with each interlock condition input.
 - Configuration of interlocks to a specific type of interlock. User Defined.

You can also configure a navigation tag for each interlock condition. If you enable navigation, the condition text on the Operator tab of the faceplate can be clicked to access the faceplate for the corresponding tag

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This code lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_IntlkAdv_4.10.00_AOIL5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Operations

This section describes the primary operations for this Add-On Instruction.

Alarms

The P_IntlkAdv Add-On Instruction does not generate any alarms. The individual input conditions can be alarmed, if necessary, in other logic before they are sent to the inputs of the P_IntlkAdv instruction. In many applications, status bits from P_AIn Analog Input or P_DIn Discrete Input instructions are sent to the P_IntlkAdv inputs, and those instructions provide alarms.



ATTENTION: Status bits should be used as interlock conditions. Use Alarm bits as interlock conditions only if you intend that the interlock condition be ignored when the corresponding alarm is disabled, suppressed, or shelved.

Simulation

The P_IntlkAdv Add-On Instruction does not have a Simulation capability.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	Processing for EnableIn False (false rung) is handled by setting the summary All interlocks OK and All interlocks (cannot be bypassed) OK status outputs to false (0). The individual interlock bit status and first out outputs are left in their last state.
Powerup (prescan, first scan)	The latch and first out states of the P_Intlk Add-On Instruction are maintained through a power-down/power-up or Run/Program/Run cycle. Any commands that are received before the first scan are discarded.
Postscan (SFC transition)	No SFC postscan logic is provided.

For more information, see the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#).

Configurable Interlock Types

There are 8 Configurable Types.

Each Type has a description- by default it's a 3 letter abbreviation (for use on the HMI to show a short version), followed by a colon and then the full type description

Each type serves a slightly different function in terms of the following aspects:

1. Affects Availability
 - Types 0,2,4,6 affect availability, Types 1,3,5,7 do not
2. Must Reset (i.e. is/isn't always latched)
 - Types 0, 1, 2, 3 Must Reset, Types 4, 5, 6, 7 are configurable as to whether or not this is required
3. Bypassable (i.e. can or can't be bypassed)
 - Types 0, 1, 4, 5 are not Bypassable, Types 2, 3, 5, 6 are configurable as to whether or not this is required

Type is displayed as a first out in Val_FirstOutType- to be displayed in interlock trip message on FTAE.

Type differentiation is also useful to plant operators in calling the correct maintenance personnel on interlock trip

Types apply a Mask to Cfg_ByPassable and Cfg_Latched, forcing them accordingly. They also apply a mask to the added availability feature

Each Type has a Cfg_HasType bit. When no types are enabled from these bits, set all types to 6- the default type(general) which has all options available

Availability

Availability has been added to the interlock bank.

There is an input availability field added for Cascading of Interlock Banks, otherwise set to True.

Availability Mask is Applied to the interlocks (Either Sts_or NB) to determine whether Sts_Avail is on or not, telling another control module whether the device in question is available for control, or not Availability mask is based on configured type, per interlock.

Programming Example

This example uses the P_Intlk instruction to concentrate the interlock conditions that allow the functioning of the refiner plates that are used for grinding wood as part of the pulp manufacturing process.

Perform the following steps to import an Add-On Instruction to your project.

1. Right-click 'Add-On Instructions' and select 'Import Add-On Instruction...'
2. On the Import Add-On Instruction dialog box, select the P_Intlk instruction and click Import.
3. Add the P_Intlk instruction to your project:
 - a. Click the Add-On tab on the Language Element toolbar.
 - b. Click the P_Intlk instruction.
 - c. Also add the P_Motor instruction that controls the refiner plates.
4. Double-click the interlock instruction name and create the tag name for it. The naming convention that makes navigation from the motor faceplate work automatically is to use the motor's tag name followed by _Intlk.
5. Create the input references necessary to ensure the appropriate operation of the refiner plates and create their appropriate tags.
6. Expose the Sts_BypActive pin on P_Motor. Wire this pin back to Inp_BypActive on P_Intlk, and mark this wire 'Assume Data Available.'

- On the P_Motor faceplate, click the HMI configuration tab and go to page 2.

The Interlock button is enabled.

14. On the P_Motor faceplate, click the Interlock button to open the P_Intlk faceplate. On the P_Intlk faceplate, click the Engineering tab, and perform the following:
 - a. Name the interlocks accordingly.
 - b. Configure interlock Type 0 - 7 accordingly.
 - c. Select the appropriate state under 'OK State'.
 - d. Select the appropriate Intlk Type.
 - e. If Intlk Type 6 or 7 selected:
Indicate which interlocks can be bypassed.
Indicate which interlocks must be reset.

TIP Specific inputs can be bypassed on the maintenance tab based on the selections.
When bypass is enabled, it bypasses only those set on the Maintenance tab.

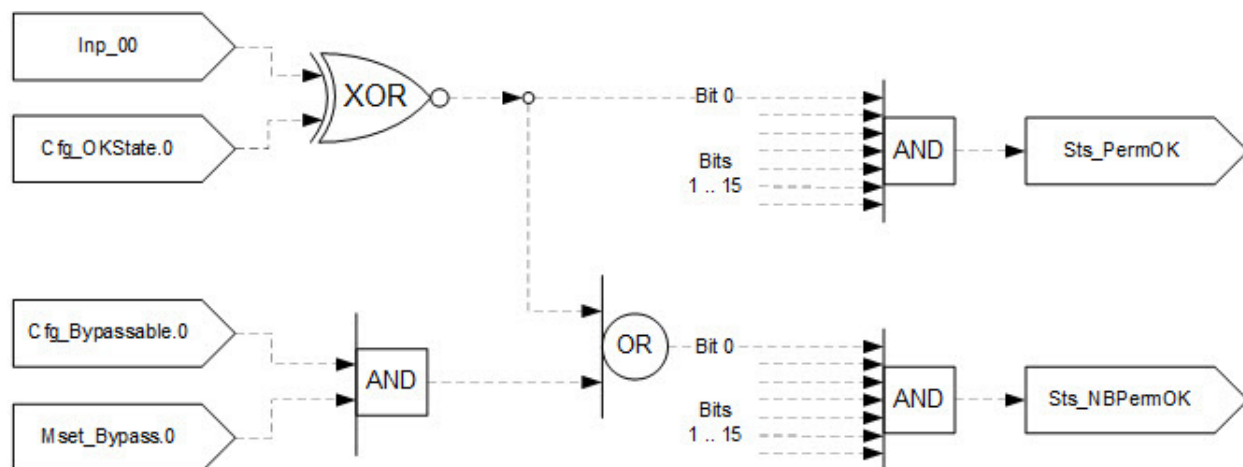
15. Connect Sts_Avail from P_IntlkAdv instruction to P_Motor instruction pin Inp_IntlkAvail.

Permissives with Bypass (P_Perm)

This section is for the operation of the Add-on Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_Perm (Permissives with Bypass) Add-On Instruction is used to collect (sum up) the permissive conditions that allow a piece of equipment to start (run, energize, open, and so forth). Permissive conditions generally must be true to start the equipment. Once the equipment is running, permissives are ignored. Use the P_Intlk (Interlocks) Add-On Instruction to collect conditions that stop running equipment and prevent it from starting.

The following diagram shows the functional characteristics of the P_Perm Add-On Instruction.



Functional Description

The P_Perm instruction provides the following capabilities:

- **Permissive input OK Check:** Evaluate the inputs; if they are all in their configured OK state, set the All Permissives OK status to true.
- **Permissive Bypass:** Evaluate the inputs that are configured as permissives that cannot be bypass or can be bypassed ($Cfg_Bypassable.x = 1$) and are set to be bypassed ($MSet_Bypass.x = 1$). If those conditions are in their configured OK state, set the 'All Non-Bypassable Permissives OK' status to true.

- **Summary Status:** The P_Perm Add-On Instruction summarizes its 16-permissive input conditions into two primary status bits: Sts_PermOK and Sts_NBPermOK. Sts_PermOK indicates that all permissive conditions are clear regardless of bypass state (ready to run regardless of bypassed state). Sts_NBPermOK indicates that all permissive conditions that cannot be bypassed are clear and all permissive conditions that are able to be bypassed are either clear or bypassed (ready to run if permissives are bypassed).

IMPORTANT Whether permissives are bypassed is determined by the downstream equipment instructions. The P_Perm instruction simply provides the two summary status bits. These 2 bits are to be wired or mapped to the equipment control logic.

- **Faceplate:** The P_Perm Add-On Instruction faceplate displays the permissive condition state of each input and whether it is bypassed, and shows the overall permissive (summary) status. The Engineering tab of the faceplate lets you configure the instruction for OK state configuration, configuration of permissives that can be bypassed. Text is associated with each permissive condition input. You can also configure a navigation tag for each permissive condition. If you enable navigation, the condition text on the Operator tab of the faceplate can be clicked to open the faceplate for that corresponding tag.

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This code lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_Perm_4.10.00_AOI.L5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Operations

This section describes the primary operations for this Add-On Instruction.

Alarms

The P_Perm Add-On Instruction does not generate any alarms. The individual input conditions can be alarmed, if necessary, in other logic before they are sent to the inputs of the P_Perm instruction. In many applications, status bits from P_AIn Analog Input or P_DIn Discrete Input instructions are sent to the P_Perm inputs.



ATTENTION: Status bits should be used as permissive conditions. Use alarm bits as permissive conditions only if you intend that the permissive condition be ignored when the corresponding alarm is disabled, shelved, or suppressed.

Simulation

The P_Perm Add-On Instruction does not have Simulation capability.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung) Handling	Processing for EnableIn False (false rung) is handled by setting the summary All Permissives OK and All Permissives (that cannot be bypassed) OK status outputs to false (0).
Powerup/Prescan Handling (initial modes)	No prescan logic is provided.
Postscan (SFC transition) Handling	No SFC postscan logic is provided.

For more information, see the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#).

Programming Example

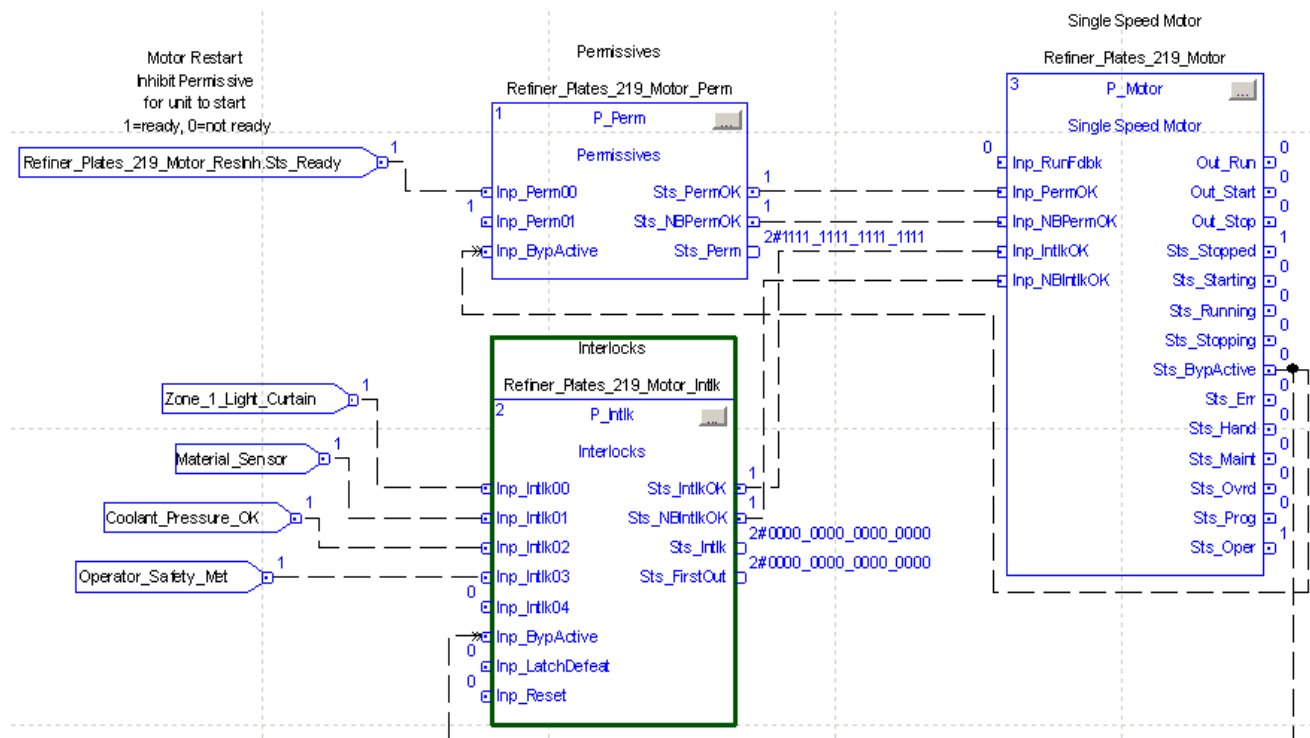
This example uses the P_Perm instruction to concentrate the permissive conditions that allow the start of a conveyor single speed motor.

Use these steps to import an Add-On Instruction to your project.

1. Right-click on “Add-On Instructions” and select “Import Add-On Instruction”
2. On the Import Add-On Instruction dialog box, select the P_Perm instruction and click Import.

3. Add the P_Perm instruction to your project by clicking the Add-On tab on the Language Element toolbar and then clicking the P_Perm instruction. Also add the P_Motor instruction that controls the extracted material conveyor.
4. Double-click the interlock instruction name and create the tag name for it. The naming convention that makes this instruction work automatically is to use the motor's tag name followed by _Perm.
5. Create the input references necessary to ensure the appropriate operation of the refiner plates and create their appropriate tags.
6. Expose the Sts_BypActive pin on P_Motor. Wire this pin back to Inp_BypActive on P_Perm, and mark this wire 'Assume Data Available'.

The following image is what the instruction looks like when connected correctly.



7. Save then download and run your Logix application.
8. In your HMI application, add a P_Motor motor graphic object from P_Motor Graphics Library.ggfx and right-click on it.
9. From the list, select “Global Object Parameter Values”
10. Associate the created tag for P_Motor of the Refiner Plates in the controller with the graphic object in the HMI:
 - a. On the first line of the Global Object Parameter Value table, type its name.
 - b. On the second line, type the name of the shortcut to the controller enclosed by brackets.
 - c. Fill in additional parameters as desired.

Because of the naming convention, the connection will be made automatically and the permissive settings will be associated with the Main Conveyor motor.

11. Save and run your HMI application.
12. Run your HMI application and click the motor graphic object. If the Quick display appears, click the 'go to faceplate' button. On the P_Motor faceplate, click the HMI Configuration tab, check 'Enable navigation to permissive object'.

After doing this, the button to access the permissive faceplate is enabled.

13. On the P_Motor faceplate, click the permissives button to open the P_Perm faceplate. On the P_Perm faceplate, select the Engineering tab, and perform the following:
 - a. Name the permissives accordingly.
 - b. Select the appropriate state under 'OK State'.
 - c. Indicate which permissives can be bypassed.
 - d. Indicate which permissives must be reset.

TIP Specific inputs can be bypassed on the P_Perm Maintenance tab.

When bypass is enabled, it bypasses only those set on the Maintenance tab.

Discrete 2-, 3-, or 4-state Device (P_D4SD)

This section is for the operation of the Add-on Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_D4SD (Discrete 2-, 3-, 4-state Device) Add-On Instruction controls and monitors feedback from a discrete 2-state, 3-state, or 4-state device in a variety of modes, monitoring for fault conditions. These devices include multiple-speed motors or multiple-position valves. The global objects and faceplate shown below are examples of the graphical interface tools that are used with this instruction.

Functional Description

The Discrete 2-, 3-, or 4-state Device Add-On Instruction provides the following capabilities:

- Provides configuration to have two, three, or four selectable states for the device.
- Provides Operator and Program commands to select one of the two, three, or four states of the device.
- Controls four discrete outputs, with configurable states of each output in the various device states. Each output can be set, cleared, or left in last state in a given device state.
- Monitors four discrete feedback inputs, with configurable states (including 'must be on', 'must be off', and 'don't care') for each input in the various device states for monitoring the actual position of the device.
- Provides configurable text labels for each of the states.
- When feedback inputs are used, detects failure to reach the target state, after a configurable time, and alarms the failure. Optionally 'sheds' to the default state (state 0) on a feedback failure.
- Monitors Permissive conditions that allow commanding the device to each state.
- Monitors Interlock conditions that return the device to its default state (state 0).
- Provides simulation of a normal working device, while holding the outputs to the real device de-energized, for use in testing or operator training.
- Monitors I/O communication status, providing an alarm on an I/O fault. Optionally 'sheds' to the default state on an I/O fault condition.
- Provides an 'Available' status when in Program command source and operating normally for use by automation logic to determine if the logic can manipulate the device.
- Operates from Hand, Maintenance, Override, External, Program, and Operator command sources.

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This code lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_D4SD_4.10.00_AOIL5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Operations

This section describes the primary operations for this Add-On Instruction.

IMPORTANT See [Appendix A](#) for Command Sources and Simulation types.

Alarms

This instruction uses the following alarms, which are implemented by using embedded P_Alarm and P_Gate Add-On Instructions.


Alarm Name	P_Alarm Name	P-Gate Name	Description
Device Fault	DeviceFault	None	Raised when the Inp_DeviceFault input is true. This alarm is provided for use by devices that generate their own fault signal. If the device fault is configured as a shed fault, the device is commanded to State 0 and a reset is required to command the device to any other state.

Alarm Name	P_Alarm Name	P-Gate Name	Description
Fail	Fail	None	Raised when the device is commanded to a new state and the device feedbacks fail to confirm that the device reached the new state within the configured time (Cfg_FailT). If the Failure is configured as a shed fault, the device is commanded to State 0 and cannot be commanded to another state until reset.
Interlock Trip	IntlkTrip	None	Raised when an interlock 'not OK' condition causes the device to transition from some other state to State 0. If interlocks are not bypassed, a bypassable interlock or a non-bypassable interlock 'not OK' condition initiates an interlock trip. If interlocks are bypassed, only a non-bypassable interlock 'not OK' condition initiates an interlock trip.
I/O Fault	IOFault	None	Raised when the Inp_IOFault input is true. This input is usually used to indicate to the instruction that a communication failure has occurred for its I/O. If the I/O Fault is configured as a shed fault, the device is commanded to State 0 and cannot be commanded to another state until reset.

Parameters of the P_Alarm object can be accessed by using the following convention: [P_Alarm Name].[P_Alarm Parameter].

Simulation

Simulation in P_D4SD de-energizes the normal outputs and simulates the feedback of a working device.

You must set the Inp_Sim parameter in the Controller to '1' to enable simulation. The Simulation or Loopback Test icon  is displayed at the top left of the Operator faceplate, indicating the device is in simulation.

While in simulation, you can set the delay (in seconds) for echoing back that the device has reached a state (Cfg_SimFdbkT).

When you have finished in simulation, set the Inp_Sim parameter in the controller to '0' to return to normal operation.

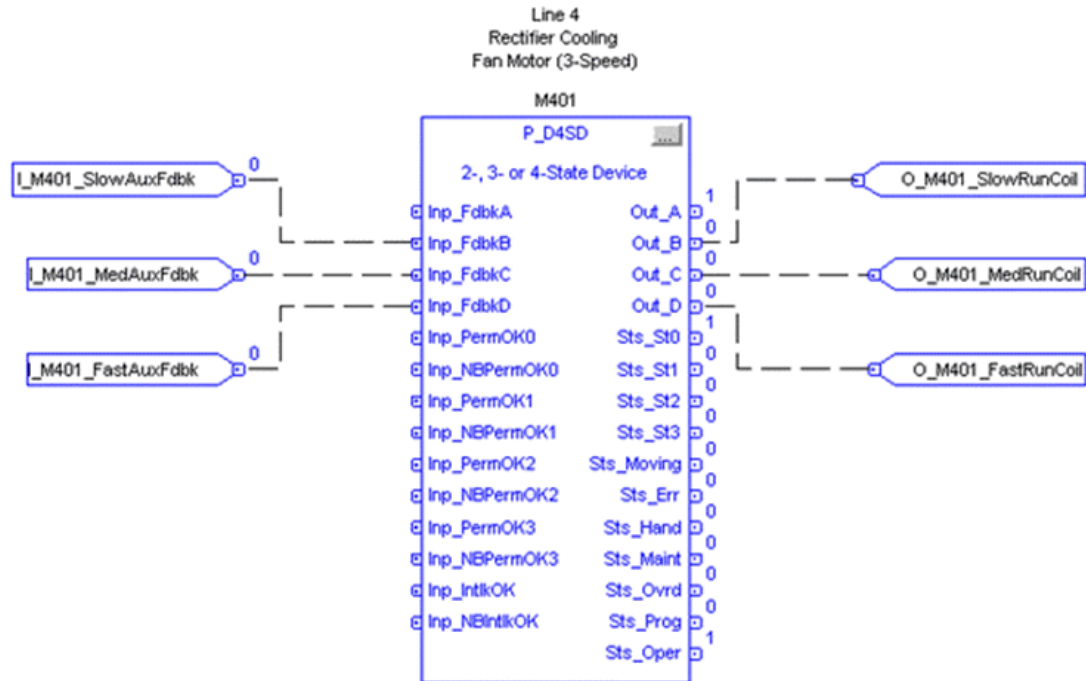
Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	Processing for EnableIn False (false rung) is handled the same as if the device were taken out of service by Command. The device outputs are de-energized and the device is shown as Program Out of Service on the HMI. All alarms are cleared.
Powerup (prescan, first scan)	On Prescan, any commands that are received before First Scan are discarded. The device is de-energized. On first scan, the device is treated as if it were returning from Hand command source: the instruction state is set based on the position feedback that is received from the device. Embedded P_CmdSrc and P_Alarm instructions are handled in accordance with their standard powerup procedures. See the P_CmdSrc and P-Alarm reference manuals for details.
Postscan	No SFC Postscan logic is provided.

Programming Example

This example uses the P_D4SD Add-On Instruction to control a cooling fan that has three fixed speeds ('low', 'medium', 'high') and an 'off' state. This is considered a 4-state device. In this example, three digital outputs are used to set the speed setting (when all three are off, the fan is commanded off) and three digital inputs provide feedback of the actual fan state (when all three are off, the fan is off).



In this example, the four cooling fan states are being mapped to the device as follows:

- State 0 = Off
- State 1 = Low
- State 2 = Medium
- State 3 = High

Set the Cfg_NumStates parameter to 4 to indicate this is a four-state device. The input parameters for states 1...3 (Inp_FdbkB, Inp_FdbkC, Inp_FdbkD) are connected to the digital inputs, representing the status of the fan. The output parameters for states 1...3 (Out_B, Out_C, Out_D) are connected to the digital outputs that command the fan to the desired state.

Based on the wiring of the I/O, we can now configure the P_D4SD instruction how we want to process the outputs to get to the desired state. We can do this via the following table.

Table 20 - P-D4SD Example Outputs

	Output A	Output B	Output C	Output D
State 0	1	0	0	0
State 1	0	1	0	0
State 2	0	0	1	0
State 3	0	0	0	1

1 = command output On, 0 = command output Off.

We are setting Output A so it can be used for display purposes even though Output A is not used by the cooling fan device. The parameter Cfg_OutSt[x]Write determines which outputs get written for each state. The parameter Cfg_OutSt[x]State determines the state that gets written. These parameters are single integers where bit 0 represents output A and bit 3 represents output D.

These parameters are displayed in binary format as indicated by the prefix 2#. By using [Table 20](#), we can set the settings as follows:

```

Cfg_OutSt0Write:    2#0000_1111
Cfg_OutSt0State:    2#0000_0001
Cfg_OutSt1Write:    2#0000_1111
Cfg_OutSt1State:    2#0000_0010
Cfg_OutSt2Write:    2#0000_1111
Cfg_OutSt2State:    2#0000_0100
Cfg_OutSt3Write:    2#0000_1111
Cfg_OutSt3State:    2#0000_1000

```

We can now repeat this same effort to configure how the P_D4SD instruction determines actual state based on the field inputs via the following table.

Table 21 - P_D4SD Example Inputs

	Input A	Input B	Input C	Input D
State 0	x	0	0	0
State 1	x	1	0	0
State 2	x	0	1	0
State 3	x	0	0	1

x = status not checked, 1 = status checked on, 0 = status checked off

The parameter Cfg_FdbkSt[x]Check determines which feedback inputs to check for each state. The parameter Cfg_FdbkSt[x]State determines how the state is interpreted from the input values.

By using [Table 21](#), we can set the settings as follows:

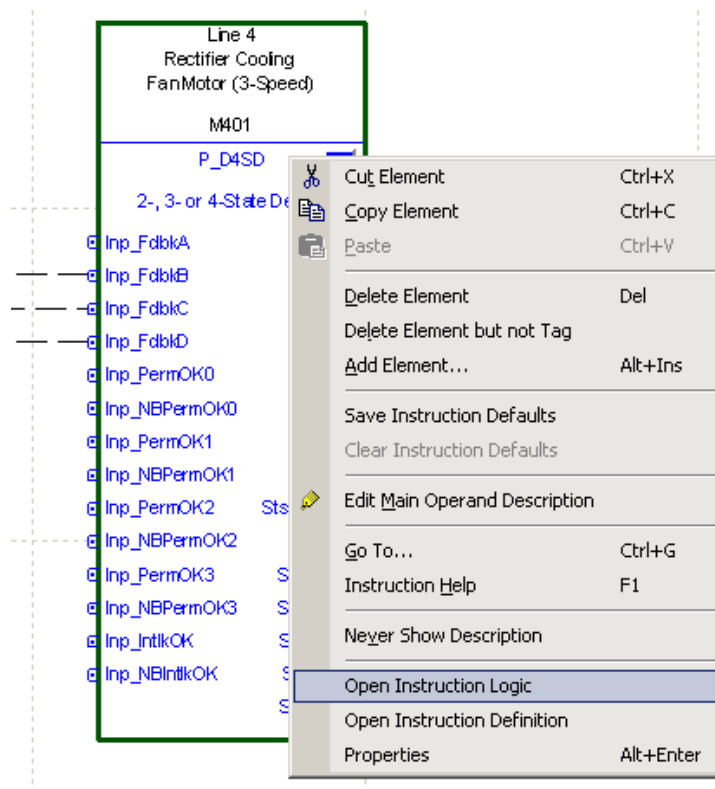
Cfg_FdbkSt0Check:	2#0000_1110
Cfg_FdbkSt0State:	2#0000_0000
Cfg_FdbkSt1Check:	2#0000_1110
Cfg_FdbkSt1State:	2#0000_0010
Cfg_FdbkSt2Check:	2#0000_1110
Cfg_FdbkSt2State:	2#0000_0100
Cfg_FdbkSt3Check:	2#0000_1110
Cfg_FdbkSt3State:	2#0000_1000

As this is a cooling fan, if there is a device mismatch or fault, we still want the logic to command to the desired state. Therefore, Cfg_ShedOnFail and Cfg_ShedOnDeviceFault are both set to 0.

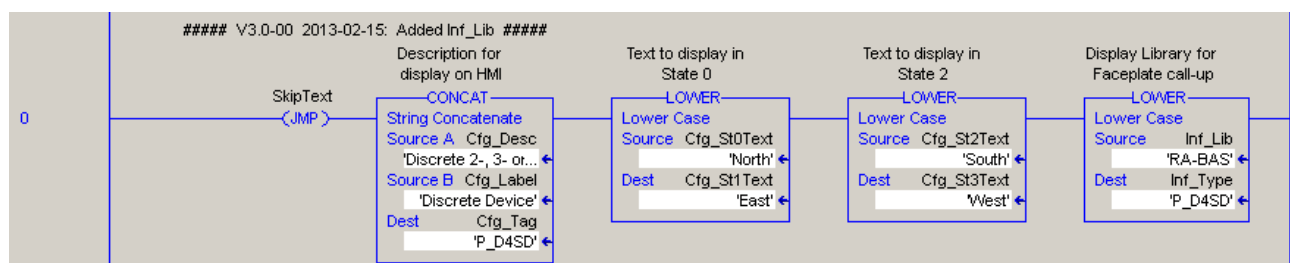
Lastly, configure the following local configuration tags to drive the text on the operations faceplate. In this example, the cooling fan P&ID tag is M401. In this example, they are set as follows:

Cfg_Tag:	'M401'
Cfg_Label:	'Line 4 Rectifier Fan'
Cfg_Desc:	'Line 4 Rectifier Cooling Fan'
Cfg_St0Text:	'Stopped'
Cfg_St1Text:	'Slow'
Cfg_St2Text:	'Medium'
Cfg_St3Text:	'Fast'

Local tags can be configured through the HMI faceplates or in Logix Designer application by opening the Instruction Logic of the Add-On Instruction instance and then selecting the string on the displayed rung.



All of the strings in local tags are shown on the first rung of the Add-On Instruction's 'Logic' routine for your convenience.



Central Reset (P_Reset)

This section is for the operation of the Add-on Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_Reset (Central Reset) Add-On Instruction provides a central point for resetting equipment faults. Latched alarms can be reset for a control strategy.

The P_Reset instruction accepts an Operator Reset command, a Program Reset command, and a Reset Input that can come from a push button, from a higher-level P_Reset instruction (from a containing control strategy, such as a Unit Reset sent to a P_Reset at Equipment Module scope), or from any other source.

The P_Reset instruction also includes a Reset Required input for collecting the Ready to Reset outputs of the various instructions it resets and providing a Ready to Reset (Reset Required) status that can illuminate a push button or make an HMI Reset button visible.

The P_Reset instruction includes a timer function that causes its output to be held on for at least a minimum time. This lets the reset signal be sent via physical output cards to field devices that require it (for example, motor drives) and gives time for the cleared status from the device to propagate back to Interlock or Permissive inputs.

Functional Description

The P_Reset Add-On Instruction provides the following capabilities:

- A Reset output for use by other instructions in a control strategy to reset latched alarms (for example, P_AIn analog input alarms) or equipment faults (for example, P_Motor motor faults)
- An Operator Reset command for use by the HMI
- A Program Reset command for use by control strategies
- A Reset input for use by push button inputs or for cascading resets from higher levels of the control hierarchy
- A Reset Required input and Ready for Reset output for use in highlighting to operators where a reset is required before the equipment restart
- A Minimum On Time for the reset output to allow physical equipment to clear faults and have the clear status propagate through the various permissive and interlock instructions in the control strategy

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This code lets you create your own instruction set for programming logic as a supplement to the instruction set

provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_Reset_4.10.00_AOIL5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Operations

This section describes the primary operations for this Add-On Instruction.

Alarms

The P_Reset instruction provides no alarms. However, the instruction is used to fan out a reset to instructions that have latched alarms.

Simulation

The P_Reset Add-On Instruction does not have a Simulation capability.

Execution

The following table explains the handling of instruction execution conditions.

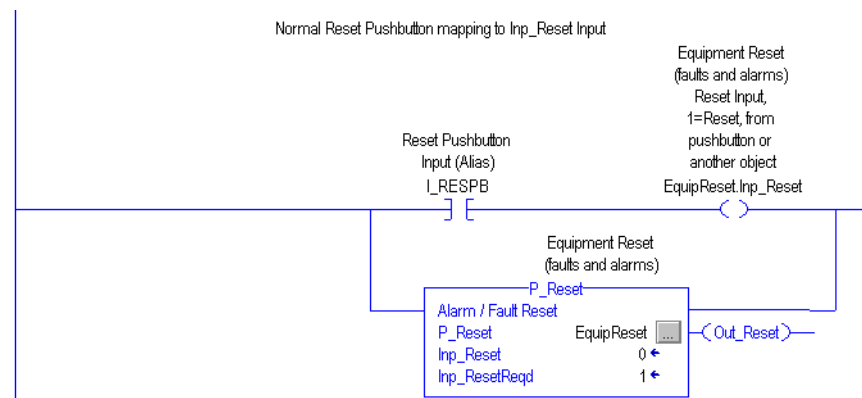
Condition	Description
EnableIn False (false rung) Handling	<p>Creates a low-to-high transition detection scheme. When the instruction is again scanned as True, the transition is detected and the reset output pulses for the required time. Otherwise, EnableIn False processing is identical to normal true rung processing: the Operator Reset command, Program Reset command, and Reset Input function as in normal processing.</p> <p>This scheme lets the Reset instruction participate in a control hierarchy by using the Inp_Reset input, yet be tied to a reset push button using the rung state.</p>

For more information, see the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#).

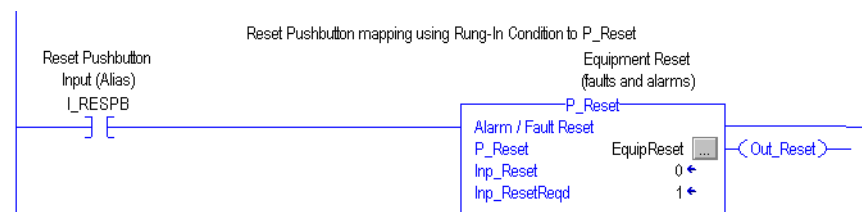
Implementation by Using EnableIn False Feature

For the convenience of Ladder Diagram programmers, the P_Reset instruction can be used in a Ladder Diagram routine with the Input condition carried by the Rung-In condition instead of being mapped on a separate branch.

The following illustration shows normal implementation with the input condition mapped to Inp_Reset on a separate branch.



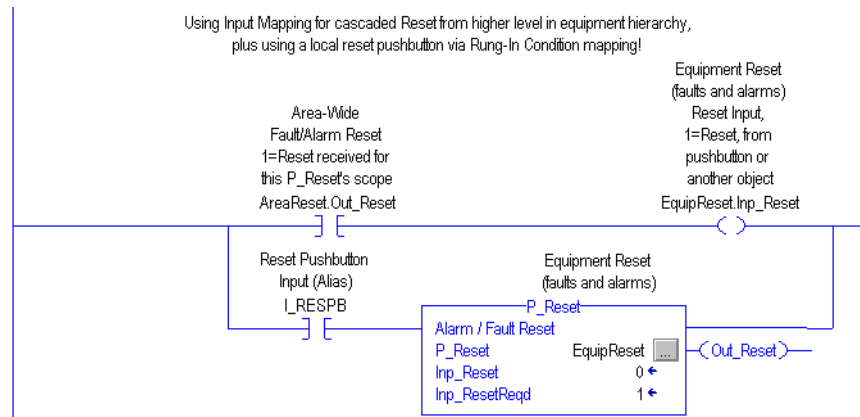
The following illustration shows EnableIn False implementation with the input condition mapped to the P_Reset instruction by using the Rung-In state.



The Rung-In condition determines whether the P_Reset instruction's normal code (Logic Routine) is performed or its EnableIn False code (EnableInFalse Routine) is performed. In the P_Reset instruction, the EnableIn False code is identical to the Logic code, except it also arms a one-shot for the Rung-In condition. When the rung goes **True**, the Logic routine one-shot fires and the Out_Reset output pulses for the configured time, executing a reset.

Use both mapping methods together to provide an additional input to the P_Reset instruction. This is especially useful in an equipment hierarchy, such as in Batch applications (Control Module, Equipment Module, Unit, Cell, Area, and so forth). The Inp_Reset input can be mapped from a higher-level P_Reset instruction, while the local Reset Pushbutton can be mapped by using the Rung-In condition to execute a local equipment reset.

The following illustration shows an EnableIn dual implementation consisting of a high-level reset by using a mapping branch and a local Reset button by using Rung-In state mapping.

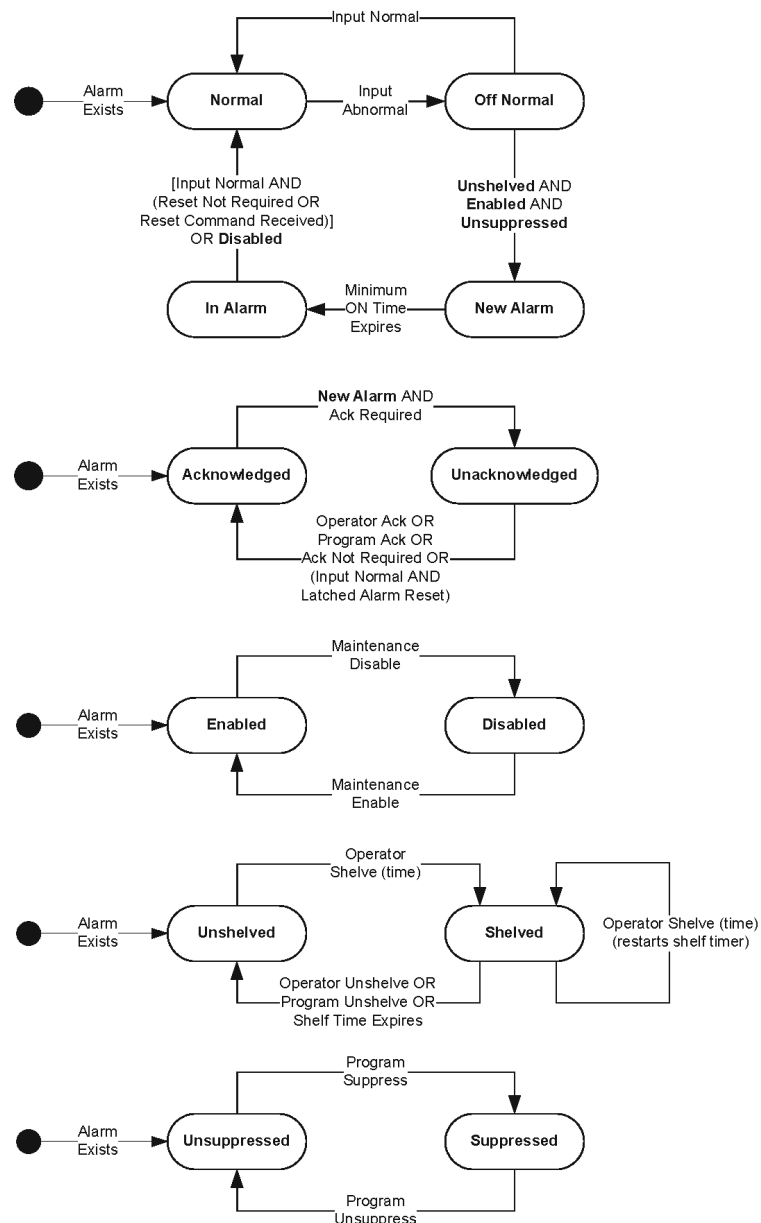


Common Alarm Block (P_Alarm)

This section is for the operation of the Add-on Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_Alarm (Common Alarm Block) Add-On Instruction is used to provide notification to operators of abnormal conditions or events. This instruction handles Alarm Acknowledgment, Alarm Reset, Alarm Shelving/Disabling, and Alarm Suppression (for FactoryTalk Alarm and Events).

The state diagram shows how a P_Alarm instruction instance behaves as an alarm occurs, is acknowledged, clears, and is reset, depending on the instruction configuration.



Functional Description

The primary operations of the P_Alarm instruction include the following:

- Raise an alarm when the input is true.
- Make sure that the alarm stays on for a configurable minimum time or until reset, even if the input condition clears.
- Perform an alarm test when the Alarm Test button is clicked. When the Alarm Test button is clicked, the selected alarm is triggered for the minimum alarm time. The test function allows the configuration of the alarm in the alarm subsystem (FactoryTalk Alarms and Events tag alarm server) to be tested. You do not have to trigger the process condition that generates the alarm. The test also lets you verify alarm configuration values such as message, severity, color, audible alarming.
- Handle Alarm Acknowledge commands from the HMI or from other logic. The requirement for acknowledgement is configurable. If acknowledgement is required, a new alarm clears the acknowledged status and an Acknowledge command is required to set the status. If acknowledgement is not required, the alarm is automatically acknowledged.
- Handle Alarm Reset commands from the HMI or from other logic. The requirement for reset is configurable. If reset is required, the alarm Input sets the Alarm condition, and it is latched in until the alarm Input is clear and a Reset command is received. If reset is not required, the Alarm condition clears when the input clears and the minimum alarm on time expires.

TIP An Add-On Instruction that contains one or more embedded P_Alarm instances provides a Reset command that is forwarded to the contained P_Alarm instances. This same reset command can also be used to clear latched fault conditions or otherwise reset the containing instruction.

- Handle Maintenance Disable and Enable commands, Program Suppress and Unsuppress commands, and Operator Shelve and Unshelve commands. Providing separate commands and status for these functions lets automatic logic suppress an alarm at certain operating sequence points. Maintenance personnel can independently disable the alarm or the operator can temporarily shelve the alarm. When the operating sequence unsuppresses the alarm at the appropriate step, the Maintenance Disable or Operator Shelve is still in effect.

IMPORTANT The P_Alarm object has output parameters that are directly written by the HMI or alarm server to be compatible with FactoryTalk Alarms and Events (tag-based) alarms and FactoryTalk View ME alarms. Output parameters that are directly written by the HMI are the following:

- Ack (Acknowledgment, set by the HMI when the Alarm is Acknowledged)
- Disabled (set by the HMI) to disable the Alarm, cleared to enable the Alarm
- Shelved (set by the HMI) when it is not displaying the Alarm, cleared when unshelved

For FactoryTalk Alarms and Events Tag alarms, set the HMI options to Acknowledge Required and Not Latched. (FactoryTalk View ME alarms are not configurable with those options.) The P_Alarm instruction handles automatic acknowledgement (Cfg_AckReqd = 0) and latching (Cfg_ResetReqd = 1).

FactoryTalk View ME alarms are not configurable for Acknowledgment Not Required, so the P_Alarm instruction handles automatic acknowledgement when configured with Cfg_AckReqd = 0.

When an alarm is Disabled by Maintenance, the following occurs:

- The Alarm Status (Alm) clears immediately.
- If the alarm is unacknowledged, it must still be acknowledged.

When an alarm is Shelved by Operator or Suppressed by Program, the following occurs:

- The alarm is not cleared until the input condition clears.
- New alarms are not raised.
- If the alarm is latched, it must still be reset (after the input condition clears).
- If the alarm is unacknowledged, it must still be acknowledged.

When an alarm is configured not to exist by engineering, the following occurs:

- The alarm status (Alm) is cleared immediately.
- If the alarm is unacknowledged, it is immediately acknowledged.
- In preparation for being configured to exist again, the following occurs:
 - Any alarm inhibits (Disabled, Suppressed, Shelved) are cleared.
 - All Program and Operator commands are cleared (every scan).
 - All timers (shelf timer, minimum on timer) are reset.

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This code lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_Alarm_4.10.00_AOIL5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Operations

This section describes the primary operations for this Add-On Instruction.

Alarms

P_Alarm objects are often embedded within another Process Object, for example P_AIn. When embedded, the Parameters of the P_Alarm objects can be accessed by using [P_Alarm Name].[P_Alarm Parameter].

Simulation

The P_Alarm Add-On Instruction does not have a Simulation capability.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	Processing for EnableIn False (False Rung) is handled the same as the main Logic Routine except that the state of Inp (the Input) is inverted. This inversion lets the P_Alarm Add-On Instruction in a ladder diagram instance have its input mapped by using the rung condition instead of using a separate branch or rung. Set the input to 1 when using the on-rung mapping.
Powerup (prescan, first scan)	No powerup, prescan, or first scan handling is required or provided. The internal timers reset on powerup, but the outputs are retained through a power cycle or run - program - run cycle.
Postscan (SFC transition)	No SFC postscan logic is provided.

For more information, see the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#).

Standalone Versus Embedded in Other Add-On Instructions

This instruction can be used standalone, with the abnormal condition simply written or wired to the Input (Inp) pin, or it can be embedded within another Add-On Instruction to provide Alarming for some condition. For example, a Motor instruction can have P_Alarm Add-On Instructions for Failure to Start, Failure to Stop, and other conditions.

When embedded within another Add-On Instruction, the following commands and configuration parameters are wired in or aliased from the containing object:

- Inp: Alarm condition input
- Inp_Reset: Alarm reset
- PCmd_Reset: Program command to reset the alarm
- PCmd_Ack: Program command to acknowledge the alarm
- PCmd_Suppress: Program command to suppress the alarm
- PCmd_Unsuppress: Program command to unsuppress the alarm
- PCmd_Unshelve: Program command to unshelve the alarm
- Cfg_AckReqd: Acknowledge Required configuration
- Cfg_ResetReqd: Reset Required configuration
- Cfg_MinOnT: The minimum amount of time (in seconds) the alarm must be help in the alarm state (kept in the operator's view) when it occurs
- Cfg_Severity: Alarm Severity 0-250 = Low, 251-500 = Medium, 501-750 = High, 751-1000 = Urgent
- Cfg_Exists: 1 if alarm needs to be processed, 0 if alarm is not used and alarm logic needs to be skipped
- PCfg_AllowExist: 1 if alarm is allowed to exist, 0 if other configuration parameters render the alarm meaningless and it cannot occur. For example, if a motor is configured to have no run feedback, its Fail to Start and Fail To Stop alarms cannot occur and so are not allowed to exist

The following output parameters need to be wired out or aliased to status bits of the containing object to make the signals available for other logic:

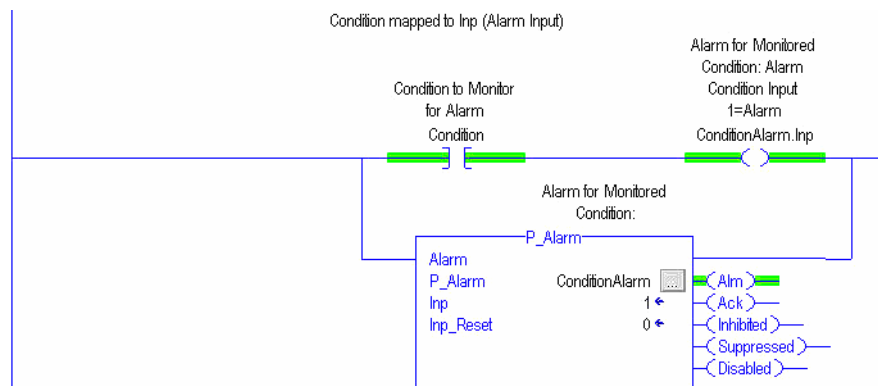
- Alm: Alarm status
- Ack: Acknowledgement status
- Disabled: Alarm Disabled status
- Shelved: Alarm Shelved status
- Suppressed: Alarm Suppressed status

IMPORTANT All of the above parameters are the targets of Alias Parameters in the containing Add-On Instruction. Acknowledge, disabled, shelved, and suppressed must be configured as read/write in RSLogix 5000 software, version 18 or later, for proper operation of alarms with the FactoryTalk Alarms and Events server tag-based alarms.

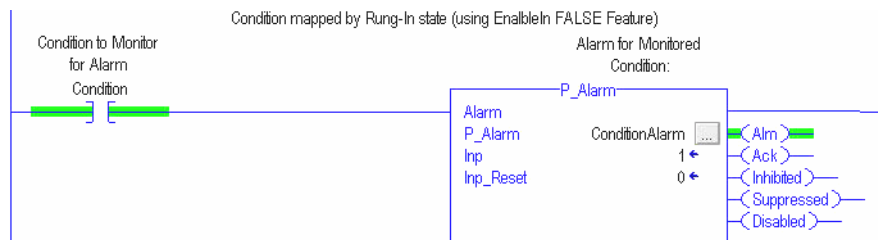
Implementation by Using the EnableIn False Feature

For the convenience of ladder diagram programmers, the P_Alarm instruction can be used in a ladder diagram routine with the input condition carried by the Rung-In condition instead of being mapped on a separate branch.

The following illustration shows normal implementation with the input condition mapped to Inp on a separate branch.



The following illustration shows the EnableIn False implementation with the input condition mapped to the P_Alarm instruction by using the Rung-In state.



The Rung-In condition determines whether the Add-On Instruction's normal code (Logic routine) is executed or its EnableIn False code (EnableInFalse routine) is executed. In the P_Alarm instruction, the EnableIn False code is identical to the Logic code, except it uses the inverse of the Inp signal for processing. To use the Rung-In mapping, method, set Inp to 1 (its default value). When the rung is True, Inp (= 1) is treated as True (not inverted, in alarm), and when the rung is False, Inp (=1) is treated as False (inverted, not in alarm).

Command Source (P_CmdSrc)

This section is for the operation of the Add-on Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_CmdSrc (Command Source) Add-On Instruction is used to provide selection of the command source (owner) of an instruction or control strategy.

Indicators on the HMI faceplate show the current states of selections for Operator, Program, External, Override, Maintenance, Hand, and Out of Service.

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_CmdSrc_4.10.00_AOI.L5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Operations

This section describes the primary operations for this Add-On Instruction.

IMPORTANT See [Appendix A](#) for Command Sources and Simulation types.

The primary operations of this instruction are the following:

- Manage the transitions between the selected sources based on user and application requests
- Enforce prioritization between sources
- Allow options to lock sources where applicable

Priorities are the following:

- Hand (highest)
- Program Out of Service
- Maintenance Out of Service
- Maintenance

- Override
- External
- Operator and Program (lowest)

Asserted inputs corresponding to methods with higher priority result in that control.

TIP Lesser priority requests are still processed and retained by the instruction, but the resultant state is that of the highest priority request.

Command Sources

The P_CmdSrc instruction uses the following standard types. The descriptions identify how types are typically used in a device whose Add-On Instruction uses P_CmdSrc.

Command Source	Description
Operator	The Operator has control of the device. Operator Command (OCmd_Xxx) and Operator Settings (OSet_Xxx) from the HMI are accepted. Operator can be Locked, which prevents Program from taking control.
Program	Program logic has control of the device. Program Commands (PCmd_Xxx) and Program Settings (PSet_Xxx) from logic are accepted. Program can be Locked, which prevents Operator from taking control.
External	An external logic source has control of the device. External Commands (XCmd_Xxx) and Settings (XSet_Xxx) from logic are accepted. External control supersedes Operator and Program when they are unlocked, and can supersede Operator Locked and Program Locked if so configured (Cfg_ExtOverLock = 1).
Override	Priority logic has control of the device. The logic supersedes Program, Operator and External sources, and can supersede Operator Locked and Program Locked if so configured (Cfg_OvrOverLock = 1). Override Commands (Inp_OvrCmd) and Override Settings (Inp_OvrXxx) from logic are accepted. If so configured (Cfg_OvrIntlkPerm = 1), bypassable interlocks and permissives are bypassed.
Maintenance	Maintenance has control of the device. Operator Commands (OCmd_Xxx) and Operator Settings (OSet_Xxx) from the HMI are accepted. Bypassable interlocks and permissives are bypassed, and device timeout checks are not processed.
Maintenance Out of Service	Maintenance has taken the device out of service. The device is disabled and outputs are held de-energized.
Programmed Out of Service	The instruction is being scanned false. The device is disabled and outputs are held de-energized (at zero). Only when Programmed Out of Service is selected are changes to the configuration of Program and Operator sources allowed. The main device Logic is not being scanned; rather, the EnableInFalse logic for the device is active.
Hand	The hardwired circuit or other logic outside the instruction controls the device. The instruction tracks the actual state of the device for bumpless transfer back to one of the other sources.

The following table shows how to request, verify, and release a command source. It also shows the input trigger style.

Command Source	Request (Input)	Acquired (Output)	Release (Input)
Hand	Inp_Hand = 1 (level)	Sts_Hand = 1	Inp_Hand = 0 (level)
Maintenance	MCmd_Acq = 1 (edge)	Sts_Maint = 1	MCmdRel = 1 (edge)
Override	Inp_Ovr = 1 (level)	Sts_Ovr = 1	Inp_Ovr = 0 (level)

Command Source	Request (Input)	Acquired (Output)	Release (Input)
External	See External Command Source	Sts_Ext=1	See External Command Source
Program	See Program Command Source	Sts_Prog = 1	See Program Command Source
Operator	See Operator Command Source	Sts_Oper = 1	See Operator Command Source

During power-up or initialization, the command source is set to the configured power-up state:

- If Cfg_ProgPwrUp = 1, the command source defaults to Program.
- If Cfg_ProgPwrUp = 0, the command source defaults to Operator.

Associated Tags

The following tables show associated tags for each command source.

Hand Command Source

Parameter	Description
Inp_Hand	1 = Hand request.
Sts_Hand	1 = Command Source is Hand.

Maintenance Command Source

Parameter	Description
MCmd_Acq	Maintenance command to acquire.
MCmd_Rel	Maintenance command to release.
Sts_Maint	1 = Command Source is Maintenance.
Sts_MAcqRcvd	1 = MCmd_Acq received this scan.

Override Command Source

Parameter	Description
Inp_Ovrd	1 = Override request.
Cfg_OvrdOverLock	1 = Override request takes control even if Program/Operator command source is locked. 0 = Override cannot take control if Program/Operator command source is locked.
Sts_Ovrd	1 = Command Source is Override.

External Command Source

Parameter	Description
XCmd_Acq	1 = External Command to Acquire (Oper/Prog to Ext)
XCmd_Rel	1 = External Command to Release (Ext to Oper/Prog)
Sts_Ext	1 = Command Source is External

IMPORTANT The external acquire and release commands can be configured as level or edge.

When CmdSrc.Cfg_ExtAcqAsLevel = 0, the commands work on EDGE.

- write 1 to XCmd_Acq to acquire
- write 1 to XCmd_Rel to release
- the commands are cleared by the instruction.

When CmdSrc.Cfg_ExtAcqAsLevel = 1, the commands work as LEVEL.

- write 1 to XCmd_Acq to acquire
- write 0 to XCmd_Acq to release
- XCmd_Rel is not used.

Operator Command Source

Parameter	Description
OCmd_Oper	Operator command to acquire from Program
OCmd_Prog	Operator command to release to Program
OCmd_Lock	Operator command to lock command source into Operator.
OCmd_Unlock	Operator command to unlock command source from Operator Locked state.
Sts_ProgOperLock	1 = Program or Operator command source is locked. If Locked, Operator cannot take from Program, Program cannot take from Operator, and Override cannot take from Program / Operator unless Cfg_OvrOverLock = 1.
Sts_ProgOperSel	1 = Program selected. 0 = Operator selected.
Sts_Oper	1 = Command Source is Operator.

Program Command Source

Parameter	Description
PCmd_Prog	Program command to acquire from Operator.
PCmd_Oper	Program command to release to Operator.
PCmd_Lock	Program command to lock command source into Program.
PCmd_Unlock	Program command to unlock command source from Program locked state.
Sts_ProgOperLock	1 = Program or Operator command source is locked. If locked, Operator cannot take from Program, Program cannot take from Operator, and Override cannot take from Program/Operator unless Cfg_OvrOverLock = 1.
Sts_ProgOperSel	1 = Program selected. 0 = Operator selected.
Sts_Prog	1 = Command Source is Program.

-
- IMPORTANT** The Program commands to acquire and release can be configured as LEVEL or EDGE.
- When Cfg_PCcmdProgAsLevel is 0, the commands work on EDGE.
- Set PCmd_Prog to 1 to acquire Program command source.
 - Set PCmd_Oper to 1 to release to Operator command source.
 - The instruction clears these parameters to 0.
- When Cfg_PCcmdProgAsLevel is 1, the commands work as LEVEL:
- Set PCmd_Prog to 1 to acquire Program command source.
 - Clear PCmd_Prog to 0 to release to Operator command source.
 - PCmd_Oper is not used.
-

IMPORTANT The Program commands to lock and unlock can be configured as LEVEL or EDGE.

When Cfg_PCmdLockAsLevel is 0, the commands work on EDGE:

- Set PCmd_Lock to 1 to acquire and lock Program command source.
- Set PCmd_Unlock to 1 to unlock Program command source.
- The instruction clears these parameters to 0.

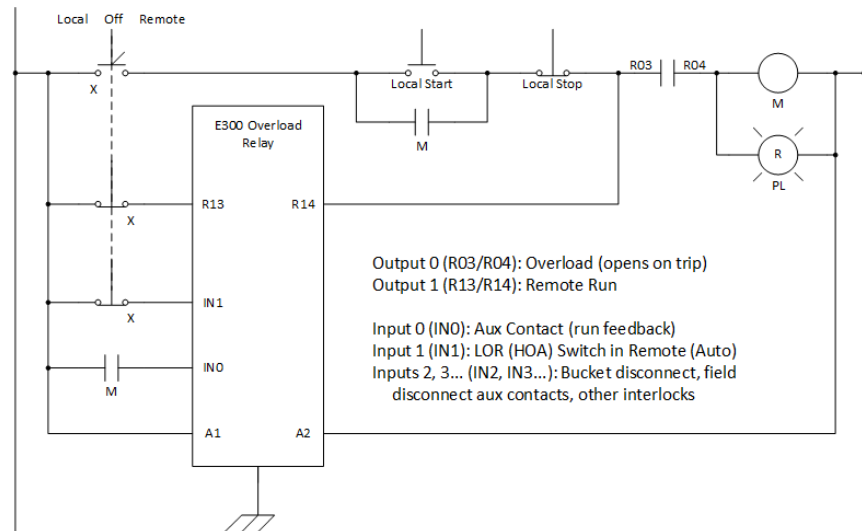
When Cfg_PCmdLockAsLevel is 1, PCmd_Lock works as LEVEL:

- Set PCmd_Lock to 1 to acquire and lock Program command source.
 - Clear PCmd_Lock to 0 to unlock Program command source.
 - PCmd_Unlock is not used.
-

Hand Command Source Example

Hand Command Source is used when a device has hardwired or other local control and the process object cannot control the object. A common example is when a hardwired Hand/Off/Auto or Local/Remote circuit is used. See [Figure 8](#). When the switch is in the Local position, CmdSrc.Inp_Hand is asserted to tell the process object it has no control of the motor. The object monitors the run feedback from the device and displays the actual state. The object also tracks the state of the device for bumpless transfer back to another Command Source (such as Operator).

Figure 8 - Hand Command Source

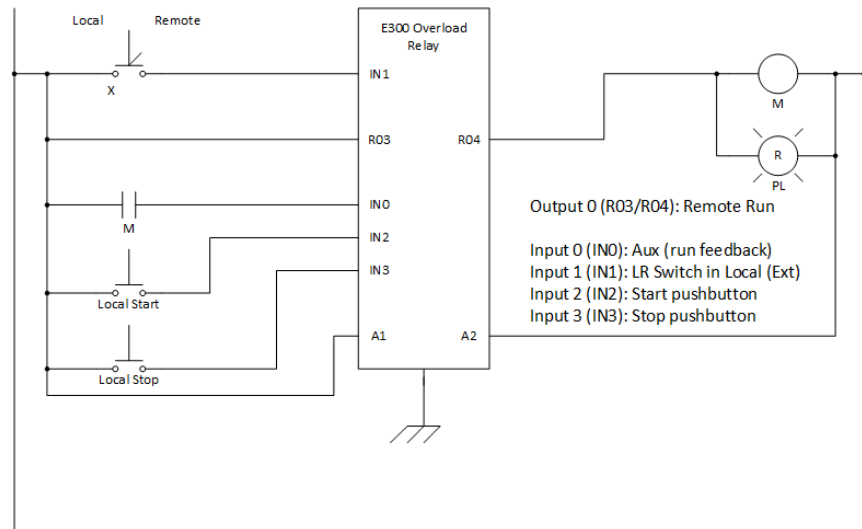


In Hand, any permissives or interlocks in the software do not apply, as the hardwired circuits ignore the outputs of the controller. Interlock conditions that apply in Hand must be wired into the circuit, in series with the starter coil. Permissive conditions that apply in Hand must be wired into the circuit in series with the Start pushbutton (if one is used).

External Command Source Example

External Command Source is used when a device has another location for control (local or remote), other than the HMI, and the process object must still control the object. A common example is when a local pushbutton operator station is used. See [Figure 9](#). The difference here is that when the switch is in the Local position, the process object is placed in External Command Source, and the process object DOES have control of the motor.

Figure 9 - External Command Source



Signals are wired as follows:

- IN0: Run Feedback to Inp_RunFdbk, 1 when motor is running
- IN1: Local / Remote switch to XCmd_Acq, 1 when switch is in Local
- IN2: Start pushbutton to XCmd_Start, 1 when button is pressed
- IN3: Stop pushbutton to XCmd_Stop, 1 when button is pressed

Set CmdSrc.Cfg_ExtAcqAsLevel to 1 to configure XCmd_Acq as level.

When XCmd_Acq is true, the faceplate shows the Command Source as External and the object starts and stops the motor normally, with commands XCmd_Start and XCmd_Stop coming from the local pushbuttons. The object obeys the normal software interlocks and permissives (and bypassing) attached to the block, because the process object is in control of the motor. Transfer to Operator, Program or another Command Source occurs when XCmd_Acq clears and is bumpless.

Alarms

The P_CmdSrc Add-On Instruction does not generate any alarms. The individual input conditions can be alarmed, if necessary, in other logic before they are sent to the inputs of the P_CmdSrc instruction.

Simulation

The P_CmdSrc Add-On Instruction does not have a Simulation capability.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	<p>If the P_CmdSrc instruction is placed on a false rung or if the EnableIn pin is set to 0, processing proceeds as normal except that these command source status bits (Sts_Hand, Sts_Maint, Sts_Ovrd, Sts_Prog and Sts_Oper) are cleared to 0.</p> <p>The command source is set to Program Out of Service (Sts_OoS = 1).</p> <p>Commands are still received for Maintenance, Operator, and Program and are processed behind the scenes, just as they are in Hand. The HMI shows the underlying requests so the Operator knows what command source is active when the instruction is again enabled.</p>
Powerup (prescan, first scan)	<p>On prescan, the Program/Operator command source selection is set based on the powerup source configuration (Cfg_ProgPwrUp):</p> <ul style="list-style-type: none"> • Cfg_ProgPwrUp= 1 Set Program/Operator selection to Program • Cfg_ProgPwrUp= 0 Set Program/Operator selection to Operator <p>The Program or Operator lock selection is set to unlocked. The Maintenance command source acquired/released state is not modified and persists through a controller powerup or PROG-to-RUN transition.</p> <p>Hand and Override command source selections are based on their Input states in the normal Logic scan; they are not modified in prescan.</p>
Postscan (SFC transition)	No SFC Postscan logic is provided.

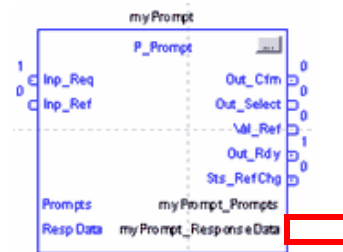
For more information, see the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#).

Operator Prompt (P_Prompt)

This section is for the operation of the Add-on Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_Prompt (Operator Prompt) Add-On Instruction is a universal mechanism for operator interaction that can be used within a control scheme. The instruction presents an operator with configurable message or data fields and accepts operator response data and confirmation.

The RespData tag at the bottom of the P_Prompt function block lets you define where to store operator responses. This tag stores any operator response as a string in the application.



Functional Description

Use a prompt to request input from an operator. The input can be any of the following:

- Acknowledging the prompt
- Viewing and confirming data
- Making a selection
- Entering numeric data
- Entering text data

Do **not** use a prompt in place of an alarm or an alert:

- An alarm, per ANSI/ISA-18.2-2016, is used to notify an operator of an abnormal situation that requires a response
- An alert is used to notify an operator of an abnormal situation that does not require a timely response

A prompt requires a response, but does not advise of an abnormal situation

	Normal Operation	Abnormal Situation
Operator Response Not Required	Normal values and status	Alert
Operator Response Required	Prompt (P_Prompt)	Alarm (P_Alarm)

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix® firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_Prompt_4.10.00_AOIL5X Add-On Instruction must be imported into the controller project to be able to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Operations

The P_Prompt Add-On Instruction does not use modes, alarms, or simulation.

IMPORTANT See [Chapter 2](#) for the P_Seq instruction. The Sequencer can be used with the Prompt instruction for manual prompt operations.

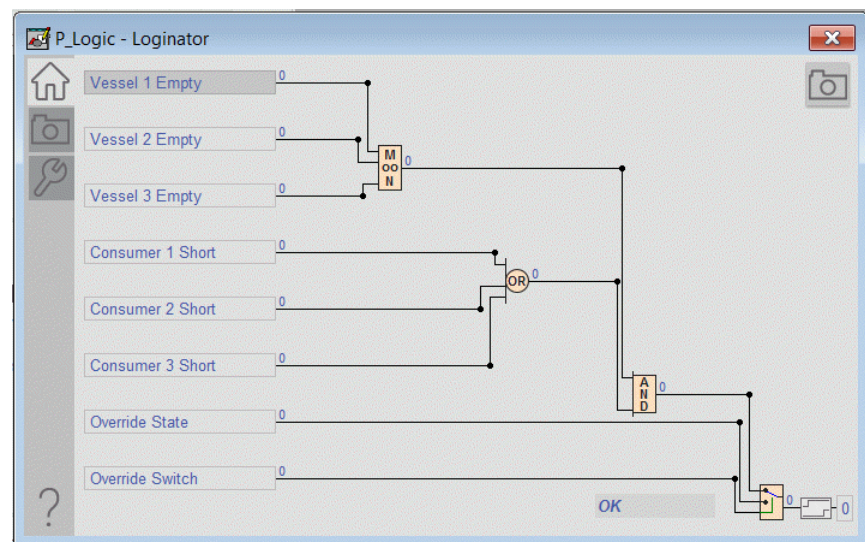
Boolean Logic with Snapshot (P_Logic)

This section is for the operation of the Add-on Instruction. For your reference, the lists of AOI parameters and local tags for each instruction family are attached to this PDF in the form of a Microsoft Excel spreadsheet. See [Access the Attachments on page 12](#) for how to access the attachments. The visualization files, display elements, global objects, and HMI information are contained in publication [PROCES-RM014](#).

The P_Logic (Boolean Logic with Snapshot) Add-On Instruction executes up to eight gates of configurable Boolean logic. Gate types available include AND, OR, XOR (Exclusive-OR), Set/Reset, Select, and Majority. Each gate provides up to four input conditions that are individually invertible. (The P_Logic instruction does not need a NOT gate.)

The P_Logic Add-On Instruction also provides a snapshot capability, enabling it to record its current state (with an optional timestamp) upon change in output state, on Operator or Program command, or based on a logic loopback input.

The following screen capture shows the functional characteristics of the P_Logic Add-On Instruction.



Functional Description

The Boolean Logic Add-On Instruction provides the following capabilities:

- Provides up to eight Boolean inputs and eight logic gates.
- Each gate has four inputs. Each input can be enabled or disabled. Each gate input can be normal or inverted. Each enabled gate input can be linked to a source, which is an instruction input or the result of a preceding gate.
- Each of the eight gates can be configured in one of the following ways:
 - Logical AND: The output of the gate is **true** if **all** enabled gate inputs (after configured inversions) are **true**. An AND gate can have 1...4 inputs enabled.

- Logical OR: The output of the gate is **true** if **any** of the enabled gate inputs (after configured inversions) are **true**. An OR gate can have 1...4 inputs enabled.
 - Logical XOR (Exclusive OR): The output of the gate is **true** if an **odd** number of the enabled gate inputs (after configured inversions) is **true**. An XOR gate can have 1...4 inputs enabled.
 - Set-Reset: The output of the gate is set **true** if one of its Set inputs is true, and is cleared to **false** if one of its Reset inputs is true. Dominant inputs have priority over non-dominant inputs.
The gate's four inputs are:
 - Input 1: SET (dominant)
 - Input 2: RESET (dominant)
 - Input 3: SET
 - Input 4: RESET
 - Select: If input 3 is **false**, the state of input 1 is passed to the gate output. If input 3 is **true**, the state of input 2 is passed to the gate output. A Select gate must have input 3 enabled and either or both of inputs 1 and 2 enabled.
 - Majority (labeled 'MooN' for 'M out of N'): The output of the gate is set **true** if most of its inputs (after configured inversions) are **true** (2 out of 2, 2 out of 3, or 3 out of 4). A Majority gate can have 2...4 inputs enabled.
- Provides configurable on-delay time and off-delay time for the output of the instruction.
 - Provides a snapshot capability, which captures the state of the instruction (all input states, gate states, and output state) for use later (until reset). The snapshot capability can be used to capture the state of the logic at the time that it tripped or shut down equipment, even if the logic states change after the shutdown. The snapshot is optionally timestamped from the controller clock (year, month, day, hour, minute, second, microsecond).
 - Provides options to enable the following snapshot trigger conditions:
 - Capture snapshot on Operator Command (OCmd_Snap).
 - Capture snapshot on Program Command (PCmd_Snap).
 - Capture snapshot when the output transitions from 0 to 1.
 - Capture snapshot when the output transitions from 1 to 0.
 - Capture snapshot of the state of the scan when a loopback input becomes **true**. This last capability enables the snapshot to be captured in a case where the P_Logic output condition was the first-out condition in a downstream P_Intlk block. The first-out indication from the P_Intlk instruction can be looped back to the Inp_Hold input of the P_Logic Instruction to hold the last-scan state in the snapshot (including time stamp of the last scan).

Required Files

Add-On Instructions are reusable code objects that contain encapsulated logic that can streamline implementing your system. This lets you create your own

instruction set for programming logic as a supplement to the instruction set provided natively in the ControlLogix firmware. An Add-On Instruction is defined once in each controller project, and can be instantiated multiple times in your application code as needed.

Controller File

The P_Logic_4.10.00_AOI.L5X Add-On Instruction must be imported into the controller project to be able to be used in the controller configuration. The service release number (boldfaced) can change as service revisions are created.

Operations

This section describes the primary operations for this P_Logic Add-On Instruction.

Configuring the Logic in a P_Logic Instance

A P_Logic instruction instance can be configured from the Logix Designer application tag monitor, but it's much easier to configure the logic from the HMI.

Rules for Set-reset Gate

The following rules apply for a Set-Reset gate:

- The dominant inputs (1 and 2) take precedence over the non-dominant (3 and 4) inputs in a Set-Reset Gate.
- If Input 1 is **true** and Input 2 is **false**, the output of the gate is Set to **true**.
- If Input 1 is **false** and Input 2 is **true**, the output of the gate is Reset to **false**.
- If both Input 1 and Input 2 are **true**, the output of the gate is not changed.
- If both Input 1 and Input 2 are **false**, Inputs 3 and 4 determine the output:
 - If Input 3 is **true** and Input 4 is **false**, the output of the gate is Set to **true**.
 - If Input 3 is **false** and Input 4 is **true**, the output of the gate is Reset to **false**.
 - If both Input 3 and Input 4 are **true**, the output of the gate is not changed.
 - If both Input 3 and Input 4 are **false**, the output of the gate is not changed.
- A Set-Reset gate must have at least one **set** input (either dominant or non-dominant) and one **reset** input (either dominant or non-dominant) enabled.

Alarms

The P_Logic Add-On Instruction does not provide any alarms. If an alarm is required, use P_Din or use the interlock alarm of the device, such as P_Motor.

One of the following applies:

- The inputs to a P_Logic instruction often come from status pins of P_Din (Discrete Input) or P_AIn (Analog Input) instructions that provide alarms for these input conditions (for example, TargetDisagree, High, Low, High-High, Low-Low).
- The output of a P_Logic instruction is typically used as an interlock condition, and the interlocked device typically provides an 'Interlock Trip' alarm.
- If an alarm is required for one of the inputs or outputs of the P_Logic instruction, a P_Alarm instruction can be added to the application logic containing the P_Logic instance.

Simulation

The Boolean Logic Add-On Instruction does not have a Simulation capability.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
Prescan	Resets the output on-delay and off-delay timers; clears the snapshot time stamp and data; clears any commands that are received while controller was in Program command source.
EnableIn False	Clears output to false (off) and resets the output on-delay and off-delay timers.
Postscan	No SFC Postscan logic is provided.

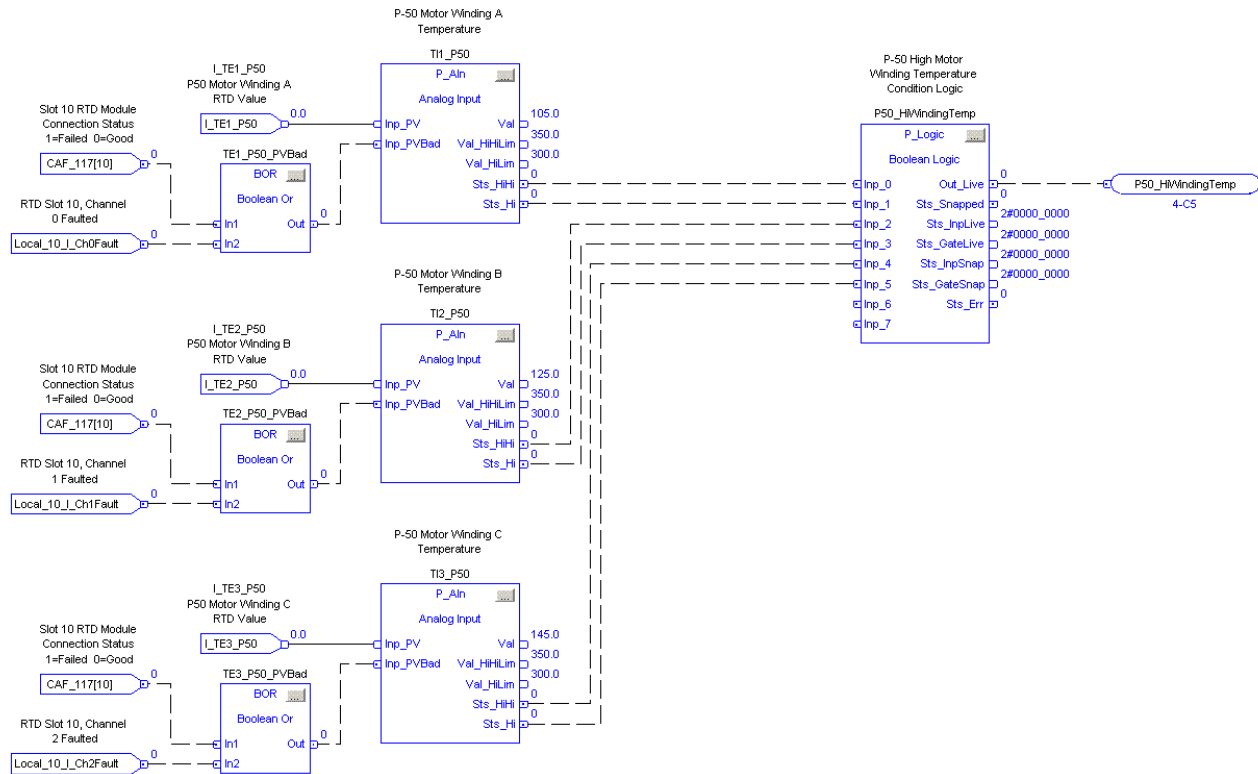
For more information, see the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#).

Programming Example

This example uses the P_Logic instruction to perform advanced interlocking logic, based on the winding temperatures of a motor. P_Logic is easier to configure through the faceplate, but this example walks through the parameter settings to fully illustrate the example.

In this example, there is a motor with three RTDs measuring temperature of the windings. To prevent damage to the windings, the motor must be interlocked if any of the three windings are above the high-high temperature limit, or if the majority of the windings are above the high temperature limit. P_Logic is being

used to perform this function. The output of this logic feeds the interlock of the motor elsewhere in logic.



The input parameters (Inp_0, Inp_1, Inp_2, Inp_3, Inp_4, Inp_5) are connected to the status outputs of the three winding temperature inputs. Three of the eight gates (0...7) in P_Logic are used in this example (1, 5, 6). Gate 1 is the OR of the three high-high status bits. Gate 5 checks if the majority of the high status bits are true. Gate 6 ORs the outputs of Gates 1 and 5 to set the output of P_Logic.

To configure the gate functions (Gates 1 and 6 as OR and Gate 5 as Majority), use the following settings:

- Cfg_GateFunc[1] = 2
- Cfg_GateFunc[5] = 6
- Cfg_GateFunc[6] = 2

Gate 1 is configured to look at the three high-high status inputs (Inp_0, Inp_2, and Inp_4) by using the following settings:

- Cfg_GateSrc1Mask.1 = 1, Cfg_GateSrc1Ptr[1] = 0
- Cfg_GateSrc2Mask.1 = 1, Cfg_GateSrc2Ptr[1] = 2
- Cfg_GateSrc3Mask.1 = 1, Cfg_GateSrc3Ptr[1] = 4

Gate 5 is configured to look at the three high status inputs (Inp_1, Inp_4, and Inp_5) by using the following settings:

- Cfg_GateSrc1Mask.5 = 1, Cfg_GateSrc1Ptr[5] = 1
- Cfg_GateSrc2Mask.5 = 1, Cfg_GateSrc2Ptr[5] = 3
- Cfg_GateSrc3Mask.5 = 1, Cfg_GateSrc3Ptr[5] = 5

Lastly, Gate 6 is configured to look at the outputs of gates 1 and 5 by using the following settings:

- Cfg_GateSrc1Mask.6 = 1, Cfg_GateSrc1Ptr[6] = 9
- Cfg_GateSrc2Mask.6 = 1, Cfg_GateSrc2Ptr[6] = 13

Cfg_OutSrcPtr needs to be set to 14 to take the output from Gate 6 and make it the output (Out_Live) of the P_Logic block.

The on-delay time is then set to 5 seconds to prevent spurious trips of the output (Cfg_OnDelay = 5).

Lastly, the string descriptions are used to provide documentation for you on the faceplate. In this example, they are set as follows:

- Cfg_0StText = OK
- Cfg_1StText = Tripped
- Cfg_Desc = Winding High Temperature Logic
- Cfg_Label = Configurable Logic
- Cfg_Tag = P_Logic
- Cfg_InpTxt[0] = Winding A Hi-Hi Temp
- Cfg_InpTxt[1] = Winding A Hi Temp
- Cfg_InpTxt[2] = Winding B Hi-Hi Temp
- Cfg_InpTxt[3] = Winding B Hi Temp
- Cfg_InpTxt[4] = Winding C Hi-Hi Temp
- Cfg_InpTxt[5] = Winding C Hi Temp

Command Source, Simulation Types

This appendix describes Command Sources, Simulation, and the specific types that apply to each object.

IMPORTANT Simulation types for Add-On Instructions are not to be confused with VMWare-style virtualization, which offers PlantPAx® system element templates.

Command Sources

The Command Source selection determines the source of Commands and Settings for the object. For example, when the Command Source is Operator, the object processes Commands and Settings from the Operator. The available Command Sources are listed in [Table 21](#).

Not all Command Sources are used in every object. [Table 22](#) provides a list of which Command Sources are used for each object.

Highlighted indicators on the object faceplate display show which sources have requested control. If more than one source is requesting control, multiple indicators are highlighted. The sources are shown in priority order, and the highlighted source furthest to the left has control. If that source relinquishes control, the next source in priority order assumes control of the object.

A triangle in the upper left corner (as seen in the following screen capture on the icon in the far right) indicates the “Normal” command source.

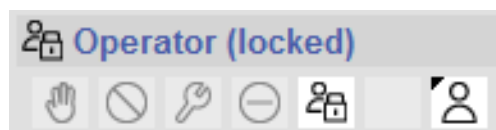


Table 21 - Command Source Descriptions








Command Source	Description
Operator 	The Operator controls the object. Operator Commands, such as OCmd_Start and OCmd_Stop, and Operator Settings, such as OSet_SP and OSet_CV, from the HMI are accepted.
Program 	Program logic controls the object. Program Commands, such as PCmd_Start and PCmd_Stop, and Program Settings, such as PSet_SP and PSet_CV, are accepted.
External 	An external system or other external devices control the object via logic. External Commands, such as XCmd_Start and XCmd_Stop, and External Settings, such as XSet_SP, XSet_CV, from Logic are accepted. Examples of external devices and systems that may control an object include a SCADA master system or local pilot devices (push buttons, switches, pilot lights).
Override 	Priority logic controls the object and supersedes Operator, Program, and External control. The Override Command Input (Inp_OvrdCmd) and other Override settings are accepted. If so configured (for example, Cfg_OvrdPermIntlk=1), bypassable interlocks and permissives are bypassed.
Maintenance 	Maintenance controls the object and supersedes Operator, Program, External, and Override control. Operator Commands and Settings from the HMI are accepted. Bypassable interlocks and permissives are bypassed, and feedback timeout checks are not processed.
Out of Service 	The object may be placed Out of Service by Maintenance from the HMI (Maintenance Out of Service). The object may also be placed Out of Service by scanning the instruction false (in a ladder diagram implementation) or by exposing and wiring the EnableIn input pin and setting it false (in a Function Block Diagram implementation). When the object is Out of Service, outputs are held de-energized / at zero, and alarms are inhibited.
Hand 	Hardwired circuits or other logic outside the instruction controls the object, ignoring outputs of the instruction. The instruction tracks the state of the object via inputs for bumpless transfer back to another command source.

Table 22 - Command Source Per Object

P_CmdSrc (1)	Operat or	Progra m	Extern al	Overri de	Maintenan ce	Out of Service	Han d
P_AOut	x	x	x	x	x	x	x
P_AOutHART	x	x	x	—	x	—	x
P_D4SD	x	x	x	x	x	x	x
P_DBC	x	x	x	x	x	x	—
P_Dose	x	x	x	—	x	—	x
P_DOut	x	x	x	x	x	x	x
P_PIDE	x	x	x	x	x	x	x
P_LLS	x	x	x	x	x	x	x
P_Motor	x	x	x	x	x	x	x
P_Motor2Spd	x	x	x	x	x	x	x
P_MotorRev	x	x	x	x	x	x	x
P_nPos	x	x	x	x	x	x	x
P_PF52x	x	x	x	x	x	x	x
P_PF6000	x	x	x	x	x	x	x
P_PF753	x	x	x	x	x	x	x
P_PF755	x	x	x	x	x	x	x
P_PF7000	x	x	x	x	x	x	x
P_SMC50	x	x	x	x	x	x	x
P_SMC FLEX	x	x	x	x	x	x	x
P_Seq	x	x	x	x	x	x	x
P_ValveC	x	x	x	x	x	x	x
P_ValveMO	x	x	x	x	x	x	x
P_ValveMP	x	x	x	x	x	x	x
P_ValveSO	x	x	x	x	x	x	x
P_VSD	x	x	x	x	x	x	x

(1) Objects that are not listed do not have any Command Source.

Simulation

There are two basic types for objects:

- Inputs (analog and discrete) — Type a simulated value into a device. For example, you want the device to work with 50 psi
- Output devices (motors, valves, drives) — Outputs to the actual device are held at zero (de-energized). The instruction behaves as if a working device were attached and operating without failures.

Notes:

Additional Add-on Instructions

Long Integer and Time Instructions

The Rockwell Automation Library of Process Objects provides additional sets of Add-On Instructions. The Logix firmware does not provide operations on Long Integers (LINT, 64-bit signed integers) used as time stamps. The instructions in [Table 23](#) provide 64-bit integer math functionality for the Library objects.

The long integer instructions are **calculation functions only**, and no HMI components are provided.

Table 23 - Long Integer Instructions

Name	Short Description	Long Description	File Name
L_ABS	Absolute Value (64 bit)	This instruction returns the absolute value (positive magnitude) of an input 64-bit integer (LINT) value.	L_ABS_1_0-00_AOI.L5X
L_ADD	Add (64 bit)	Adds two LINT (signed 64 bit) values and returns a LINT (signed 64 bit) sum. Also provides math status bits for Carry, Negative, Overflow, and Zero result (equivalent to built-in S:C, S:N, S:V, S:Z).	L_ADD_1_0-00_AOI.L5X
L_AND	Bitwise AND (64 bit)	This instruction returns the bitwise Logical AND (output bit true if both corresponding input bits true) of two input 64-bit integers (LINTs), into an output 64-bit integer.	L_AND_1_0-00_AOI.L5X
L_DEC	Decrement (64 bit)	This instruction decrements the input 64-bit integer, to return its value minus 1.	L_DEC_1_0-00_AOI.L5X
L_DIV	Divide (64 bit by 32 bit)	This instruction implements an elementary-school shift/subtract (looping) method of dividing a 64-bit integer (LINT) dividend by a 32-bit integer (DINT) divisor. The resulting quotient is a 64-bit integer (LINT), and the remainder (32-bit integer DINT) is also returned.	L_DIV_1_0-00_AOI.L5X
L_EQU	Equal (64 bit)	This instruction compares two LINT (64-bit signed integer) variables. If Inp_A is equal to Inp_B, EnableOut and Out are set to true (1). Otherwise EnableOut and Out are cleared to false (0).	L_EQU_1_0-01_AOI.L5X
L_FtoH	Float to Half-Precision	This instruction converts a 32-bit single-precision floating point number (REAL) to the best equivalent 16 bit 'half-precision' floating point number (stored in an INT, because Logix does not have a SREAL type). It accounts for positive and negative zero, subnormal (small) numbers, Infinity (+/- 1.\$), Indeterminate (-1.#IND) and Not a Number (+/- 1.#QNaN).	L_FtoH_1_0-00_AOI.L5X
L_GEQ	Greater Than or Equal (64 bit)	This instruction compares two LINT (64 -bit signed integer) variables. If Inp_A is greater than or equal to Inp_B, EnableOut and Out are set to true (1). Otherwise EnableOut and Out are cleared to false (0).	L_GEQ_1_0-01_AOI.L5X

Table 23 - Long Integer Instructions

Name	Short Description	Long Description	File Name
L_GRT	Greater Than (64 bit)	<p>This instruction compares two LINT (64-bit signed integer) variables.</p> <p>If Inp_A is greater than Inp_B, EnableOut and Out are set to true (1). Otherwise EnableOut and Out are cleared to false (0).</p>	L_GRT_1_0-01_AOI.L5X
L_HtoF	Half-Precision to Float	<p>This instruction converts a 16 bit ('half-precision') floating point number (contained in an INT, as Logix doesn't have an SREAL type) to the equivalent 32-bit single-precision floating point number (REAL).</p> <p>It accounts for positive and negative zero, subnormal (small) numbers, Infinity (+/- 1.\$), Indeterminate (-1.#IND) and Not a Number (+/- 1.#QNaN).</p>	L_HtoF_1_0-00_AOI.L5X
L_INC	Increment (64 bit)	This instruction increments the input 64-bit integer, to return its value plus 1.	L_INC_1_0-00_AOI.L5X
L_LEQ	Less Than or Equal (64 bit)	<p>This instruction compares two LINT (64-bit signed integer) variables.</p> <p>If Inp_A is less than or equal to Inp_B, EnableOut and Out are set to true (1). Otherwise EnableOut and Out are cleared to false (0).</p>	L_LEQ_1_0-01_AOI.L5X
L_LES	Less Than (64 bit)	<p>This instruction compares two LINT (64-bit signed integer) variables.</p> <p>If Inp_A is less than Inp_B, EnableOut and Out are set to true (1). Otherwise EnableOut and Out are cleared to false (0).</p>	L_LES_1_0-01_AOI.L5X
L_LIM	Limit Test (Circular) (64 bit)	<p>This instruction compares a 64-bit Input with a 64-bit High Limit and a 64-bit Low Limit.</p> <p>There are two cases: a 'normal case' (Low Limit <= High Limit) and a 'circular case' (Low Limit > High Limit) In the normal case, EnableOut, and Out are set if: Low Limit <= Input <= High Limit In the circular case, EnableOut, and Out are set if: Input >= Low Limit OR Input <= HighLimit (remember, High Limit < Low Limit)</p> <p>This instruction can be used in ladder diagram, structured text, or function block Routines just like the LIM instruction is used for 32-bit integers and floating point numbers. However, because it has InOut Parameters (references to tags of LINT type), it is not left justified on ladder rungs.</p> <p>On a False rung in LD, or in FBD with EnableIn cleared to 0, Out is cleared to 0.</p>	L_LIM_1_0-01_AOI.L5X
L_MEQ	Masked Equal (64 bit)	<p>Performs a 64-bit bitwise comparison of a Source Value against a Compare Value, and returns true if they are the same in all bit positions that have a '1' in the Mask Value. Therefore, output is true if (Source AND Mask) = (Compare AND Mask).</p> <p>On a False rung in LD, or in FBD with EnableIn cleared to 0, Out is cleared to 0.</p>	L_MEQ_1_0-01_AOI.L5X
L_MUL	Multiply (64-bit X 32-bit)	This instruction implements an elementary-school multiply-and-add-partial-products (place notation) method of multiplying a 64-bit integer (LINT) by a 32-bit integer (DINT). The resulting product is a 64-bit integer (LINT).	L_MUL_1_0-01_AOI.L5X

Table 23 - Long Integer Instructions

Name	Short Description	Long Description	File Name
L_MVM	Move with Mask (64 bit)	<p>Performs a 64 bit bitwise Move with Mask of a Source Value to an Output. If a bit in the Mask is true, the corresponding Source bit is copied to the Output. If a bit in the Mask is false, the corresponding Output bit is left unchanged.</p> <p>In other words, Output = (Output AND NOT Mask) OR (Source AND Mask).</p>	L_MVM_1_0-00_AOI.L5X
L_NEG	Negate (64 bit)	This instruction returns the negative (2's complement) of an input 64-bit integer (LINT) value.	L_NEG_1_0-00_AOI.L5X
L_NEQ	Not Equal (64 bit)	<p>This instruction compares two LINT (64-bit signed integer) variables.</p> <p>If Inp_A is not equal to Inp_B, EnableOut and Out are set to true (1). Otherwise EnableOut and Out are cleared to false (0).</p>	L_NEQ_1_0-01_AOI.L5X
L_NOT	Bitwise NOT (64 bit)	This instruction returns the bitwise Logical NOT (inverse or 1's complement) of an input 64-bit integer (LINT) value. (It flips all bits.)	L_NOT_1_0-00_AOI.L5X
L_OR	Bitwise OR (64 bit)	This instruction returns the bitwise Logical OR (output bit true if either of the corresponding input bits are true) of two input 64-bit integers (LINTs), into an output 64-bit integer.	L_OR_1_0-00_AOI.L5X
L_OTE	Output Energize (64 bit)	<p>This instruction energizes the given bit of the referenced LINT (64-bit integer) tag, that is, it sets the given bit (true, 1) if the EnableIn condition is true, or clears the given bit (false, 0) if the EnableIn condition is false.</p> <p>If the given bit number is outside the range 0...63, the controller major faults on an invalid array index (bad bit number). There is no validity check of the given bit number.</p>	L_OTE_1_0-00_AOI.L5X
L_OTL	Output Latch (64 bit)	<p>This instruction latches the given bit of the referenced LINT (64-bit integer) tag, that is, it sets the given bit (true, 1) if the EnableIn condition is true, or leaves the given bit (and the referenced LINT) unmodified if the EnableIn condition is false.</p> <p>If the given bit number is outside the range 0...63, the controller major faults on an invalid array index (bad bit number). There is no validity check of the given bit number.</p>	L_OTL_1_0-00_AOI.L5X
L_OTU	Output Unlatch (64 bit)	<p>This instruction unlatches the given Bit of the referenced LINT (64-bit integer) tag, that is, it clears the given bit (false, 0) if the EnableIn condition is true, or leaves the given bit (and the referenced LINT) unmodified if the EnableIn condition is false.</p> <p>If the given Bit number is outside the range 0...63, the controller major faults on an invalid array index (bad bit number). There is no validity check of the given bit number.</p>	L_OTU_1_0-00_AOI.L5X
L_SEL	Select (64 bit)	<p>This instruction returns Input A if the input selector bit is false, Input B if the selector bit is true.</p> <p>IMPORTANT: When EnableIn is false, the input selector bit sense is reversed. With the selector bit default value of 1, the rung state in an LD instance controls the selector in a straightforward manner. If the rung is true, select Inp_B; if the rung is false, select Inp_A.</p>	L_SEL_1_0-00_AOI.L5X
L_SUB	Subtract (64 bit)	<p>Subtracts two LINT (signed 64 bit) values and returns a LINT (signed 64 bit) difference.</p> <p>Also provides math status bits for Carry (borrow), Negative, Overflow, and Zero result (equivalent to built-in S:C, S:N, S:V, S:Z).</p>	L_SUB_1_0-00_AOI.L5X

Table 23 - Long Integer Instructions

Name	Short Description	Long Description	File Name
L_XIC	Examine On (64 bit)	<p>This instruction examines the given Bit of the input LINT (64-bit integer) and outputs true (1) if the bit is 1, false (0) if the bit is 0.</p> <p>IMPORTANT: Use the output rung state or EnableOut to feed downstream logic. The output bit 'Out' reflects only the state of the given bit, for ladder animation, and not the rung state.</p> <p>If the given bit number is outside the range 0...63, the controller major faults on an invalid array index (bad bit number). There is no validity check of the given bit number.</p>	L_XIC_1_0-00_AOI.L5X
L_XIO	Examine Off (64 bit)	<p>This instruction examines the given Bit of the input LINT (64-bit integer) and outputs true (1) if the bit is 0, false (0) if the bit is 1.</p> <p>IMPORTANT: Use the output rung state or EnableOut to feed downstream logic. The output bit 'Out' reflects only the state of the given bit, for ladder animation, and not the rung state.</p> <p>If the given bit number is outside the range 0...63, the controller major faults on an invalid array index (bad bit number). There is no validity check of the given bit number.</p>	L_XIO_1_0-00_AOI.L5X
L_XOR	Bitwise XOR (64 bit)	<p>This instruction returns the bitwise Logical XOR (exclusive OR, output bit true if either but NOT both of the corresponding input bits are true) of two input 64-bit integers (LINTs), into an output 64-bit integer.</p>	L_XOR_1_0-00_AOI.L5X
T_LtoT	LTIME to DateTime	<p>This instruction converts an LTIME (64-bit integer time stamp, for example, from an ALMD or ALMA instruction or the WallClockTime object) to a DateTime (Year, Month, Day, Hour, Minute, Second, Microsecond as DINTs) in Coordinated Universal Time (UTC offset = 0).</p> <p>The input LTIME is the 64 bit (LINT) number of microseconds since DT#1970-01-01_00:00:00.000000 UTC.</p>	T_LtoT_1_0-00_AOI.L5X
T_TtoL	DateTime to LTIME	<p>This instruction converts a DateTime (Year, Month, Day, Hour, Minute, Second, Microsecond as DINTs) in Coordinated Universal Time (UTC offset = 0) to an LTIME (64-bit integer time stamp, for example, from an ALMD or ALMA instruction or the WallClockTime object).</p> <p>The output LTIME is the 64 bit (LINT) number of microseconds since DT#1970-01-01_00:00:00.000000 UTC.</p>	T_TtoL_1_0-00_AOI.L5X

Time and Date Instructions

The Rockwell Automation Library of Process Objects also includes instructions for time and date functions.

The time and date instructions are **calculation functions only**, and no HMI components are provided.

Table 24 - Time and Date Instructions

Name	Short Description	Long Description	File Name
T_ADD	DateTime:= DateTime + Time	<p>T_ADD: Add Date/Time plus time to get new Date/Time</p> <p>This instruction adds a given amount of Time to a Date/Time to arrive at a new Date/Time. The new Date/Time is 'normalized', that is, given as a valid (if possible) Gregorian Date and Time:</p> <ul style="list-style-type: none"> • 0 <= Microseconds < 1,000,000 • 0 <= Seconds < 60 (This instruction cannot add leap seconds) • 0 <= Minutes < 60 • 0 <= Hours < 24 • 1 <= Day <= 31 and Date is a valid Gregorian date • 1 <= Month <= 12 	T_ADD_1_0-01_AOI.L5X
T_Clock	Date/Time Clock	<p>This object manages the controller 'Wall Clock', providing date and time services, including:</p> <ul style="list-style-type: none"> • Accepts downloaded date and time from HMI or other sync source and sets the clock • Reads the clock and provides the local date and time to other logic <ul style="list-style-type: none"> – IMPORTANT: Current date/time is provided as individual DINTs and as a Date Time type for use with Date/Time math instructions (T_ADD, T_SUB, T_GRT, and so forth) • Calculates and provides the day of the week for the current date <ul style="list-style-type: none"> – IMPORTANT: Use T_DoW to calculate the day of the week for any given date • Optionally sets a flag once a day to request a clock sync update • Based on configured shift start times, determines the current production shift (for up to three shifts). The controller clock can be synchronized by writing a valid year, month, day, hour, minute, and second into the appropriate settings. When the clock has been set, the settings are returned to '-1' and the time is reflected in the corresponding values and status. 	T_Clock_1_0-01_AOI.L5X
T_DIFF	Time:= DateTime - DateTime	<p>T_DIFF: Date/Time minus Date/Time gives time difference</p> <p>This instruction is given two Date/Time points and determines the amount of time between them. The result is given in days, hours, minutes, seconds, and microseconds. (Years and months are returned as zero, as the number of months is not used.) The Date/Time parameters must be valid Gregorian Dates and valid clock times:</p> <ul style="list-style-type: none"> • 0 <= Microseconds < 1,000,000 • 0 <= Seconds < 60 (This instruction cannot add leap seconds) • 0 <= Minutes < 60 • 0 <= Hours < 24 • 1 <= Day <= 31 and Date is a Valid Gregorian Date • 1 <= Month <= 12 	T_DIFF_1_0-00_AOI.L5X

Table 24 - Time and Date Instructions

Name	Short Description	Long Description	File Name
T_DoW	Day of the Week	<p>T_DoW: Day of the Week</p> <p>This instruction takes a given Date/Time, and, for the date part, returns the day of the week: (0 = Sun, 1 = Mon, 2 = Tue, 3 = Wed, 4 = Thu, 5 = Fri, 6 = Sat)</p> <p>If the given date is invalid, a flag is set (but the calculated day of the week is returned anyway.)</p> <p>IMPORTANT: The time part of input parameter DT (hours, minutes, seconds, microseconds) is ignored.</p>	T_DoW_1_0-00_AOI.L5X

Table 24 - Time and Date Instructions

Name	Short Description	Long Description	File Name
T_DST	Daylight Savings Time	<p>This instruction manages Daylight Saving Time. It uses a number of configuration values to allow handling a wide variety of national and regional rules for when to start and end Daylight Saving Time (or 'summer time').</p> <p>For use with HMI, it also provides values for display of the Month/Day and Hour/Minute of the points in time when DST starts and ends. Plus, for logging logic, it provides bits to indicate when time stamps have an overlap (1:30 a.m. happens twice) or there is a gap (one-shot).</p> <p>Follow these steps for best results.</p> <ol style="list-style-type: none"> 1. Clear Cfg_EnableDST to 0. 2. Open the Controller Properties, clear the DST checkbox, and set the clock to local STANDARD time 3. Configure the T_DST instruction per the following instructions. 4. Set the Cfg_EnableDST bit to 1. <p>The clock is switched to DST based on the rules that are entered if DST is in effect for your location.</p> <p>Configuration:</p> <ul style="list-style-type: none"> • Cfg_FwdMo -- Month that is specified in rule for date to spring forward (1...12) • Cfg_FwdOccur -- Occurrence of day of week to spring forward 1 = first, 2 = second...5 = last • Cfg_FwdDoW -- Day of the week to spring forward (0 = Sun...6 = Sat) • Cfg_FwdDoM -- Day of month for spring forward if on a fixed date (1...31) • Cfg_FwdDoWBefore -- Day of the week BEFORE the first...last day of week or date (0 = Sun...6 = Sat) • Cfg_FwdHr --Hour (LOCAL) to spring forward (0...23) • Cfg_FwdMin -- Minute (LOCAL) to spring forward (0...59) • Cfg_FwdFixedDate --1 = Spring forward on fixed date, 0 = on occurrence of day of week • Cfg_FwdUseBefore --1 = Spring forward on day of week before date or day of week • Cfg_BackMo -- Month that is specified in rule for date to fall back (1...12) • Cfg_BackOccur -- Occurrence of day of week to fall back 1 = first, 2 = second...5 = last • Cfg_BackDoW -- Day of the week to fall back (0 = Sun...6 = Sat) • Cfg_BackDoM -- Day of month for fall-back if on a fixed date (1...31) • Cfg_BackDoWBefore -- Day of the week BEFORE the first...last day of week or date (0 = Sun...6 = Sat) • Cfg_BackHr --Hour (LOCAL) to fall back (0...23) • Cfg_BackMin -- Minute (LOCAL) to fall back (0...59) • Cfg_BackFixedDate -- 1 = Fall-back on fixed date, 0=on occurrence of day of week • Cfg_BackUseBefore: --1 = Fall-back on day of week before date or day of week • Cfg_Offset --Number of minutes to spring forward or fall back (0...1439, default = 60) • Cfg_EnableDST --1 = Automatically adjust clock for DST, 0 = Always Standard Time, no DST 	T_DST_1_0-01_AOI.L5X

Configuration Values for T_DST for Sample Rulesets

(T_DST Was Tested in Each of These Configurations)

"Fall Back" Rule	First Sunday in November at 02:00 Local	Last Sunday in October at 01:00 UTC	Saturday before Last Sunday in October at 23:00 UTC	For 2014: June 28 at 02:00 Local	Sunday between Rosh Hashanah and Yom Kippur (varies)
Cfg_BackMo	11	10	10	6	varies
Cfg_BackOccur	1	5	5	---	varies
Cfg_BackDoW	0	0	0	---	0
Cfg_BackDoM	-	-	-	28	-
Cfg_BackDoWBefore	-	-	6	-	-
Cfg_BackHr	2	varies by zone	varies by zone	2	2
Cfg_BackMin	0	0	0	0	0
Cfg_BackFixedDate	0 (false)	0 (false)	0 (false)	1 (true)	0 (false)
Cfg_BackUseBefore	0 (false)	0 (false)	1 (true)	0 (false)	0 (false)
Cfg_Offset	60	60	60	60	60
Cfg_EnableDST	1	1	1	1	1

Name	Short Description	Long Description	File Name
T_EQU	DateTime = DateTime?	<p>This instruction compares two Date-and-Time-of-Day (DateTime) variables.</p> <p>If DT1 is equal to DT2, EnableOut and Out are set to true (1). Otherwise EnableOut and Out are cleared to false (0).</p> <p>This instruction can be used in ladder diagram, structured text, or function block Routines just like the EQU instruction is used for integers and floating point numbers. However, because it has InOut Parameters (tag references to user-defined data types), it is not left justified on ladder rungs.</p> <p>On a False rung in LD, or in FBD with EnableIn cleared to 0, Out is cleared to 0.</p> <p>IMPORTANT: This instruction is dependent on the user-defined data type 'DateTime' (external to this Add-On Instruction definition).</p> <p>'DateTime' is defined as the following:</p> <ul style="list-style-type: none"> • Yr -- DINT (year) • Mo -- DINT (month) • Da -- DINT (day) • Hr -- DINT (hour) • Min -- DINT (minute) • Sec -- DINT (second) • uSec -- DINT (microsecond) 	T_EQU_1_0-01_AOI.L5X

Name	Short Description	Long Description	File Name
T_GEQ	DateTime >= DateTime?	<p>This instruction compares two Date-and-Time-of-Day (DateTime) variables.</p> <p>If DT1 is greater than (after) or equal to DT2, EnableOut and Out are set to true (1). Otherwise EnableOut and Out are cleared to false (0).</p> <p>This instruction can be used in ladder diagram, structured text, or function block Routines just like the GEQ instruction is used for integers and floating point numbers. However, because it has InOut Parameters (tag references to user-defined data types), it is not left justified on ladder rungs.</p> <p>On a False rung in LD, or in FBD with EnableIn cleared to 0, Out is cleared to 0.</p> <p>IMPORTANT: This instruction is dependent on the user-defined data type 'DateTime' (external to this Add-On Instruction definition).</p> <p>'DateTime' is defined as the following:</p> <ul style="list-style-type: none"> • Yr -- DINT (year) • Mo -- DINT (month) • Da -- DINT day) • Hr -- DINT (hour) • Min -- DINT (minute) • Sec -- DINT (second) • uSec -- DINT (microsecond) 	T_GEQ_1_0-01_AOI.L5X
T_GRT	DateTime > DateTime?	<p>This instruction compares two Date-and-Time-of-Day (DateTime) variables.</p> <p>If DT1 is greater than (after) DT2, EnableOut and Out are set to true (1). Otherwise EnableOut and Out are cleared to false (0).</p> <p>This instruction can be used in ladder diagram, structured text, or function block Routines just like the GRT instruction is used for integers and floating point numbers. However, because it has InOut Parameters (tag references to user-defined data types), it is not left justified on ladder rungs.</p> <p>On a False rung in LD, or in FBD with EnableIn cleared to 0, Out is cleared to 0.</p> <p>IMPORTANT: This instruction is dependent on the user-defined data type 'DateTime' (external to this Add-On Instruction definition).</p> <p>'DateTime' is defined as the following:</p> <ul style="list-style-type: none"> • Yr -- DINT (year) • Mo -- DINT (month) • Da -- DINT day) • Hr -- DINT (hour) • Min -- DINT (minute) • Sec -- DINT (second) • uSec -- DINT (microsecond) 	T_GRT_1_0-01_AOI.L5X

Name	Short Description	Long Description	File Name
T_LEQ	DateTime <= DateTime?	<p>This instruction compares two Date-and-Time-of-Day (DateTime) variables.</p> <p>If DT1 is less than (before) or equal to DT2, EnableOut and Out are set to true (1). Otherwise EnableOut and Out are cleared to false (0).</p> <p>This instruction can be used in ladder diagram, structured text, or function block Routines just like the LEQ instruction is used for integers and floating point numbers. However, because it has InOut Parameters (tag references to user-defined data types), it is not left justified on ladder rungs.</p> <p>On a False rung in LD, or in FBD with EnableIn cleared to 0, Out is cleared to 0.</p> <p>IMPORTANT: This instruction is dependent on the user-defined data type 'DateTime' (external to this Add-On Instruction definition).</p> <p>'DateTime' is defined as the following:</p> <ul style="list-style-type: none"> • Yr -- DINT (year) • Mo -- DINT (month) • Da -- DINT day) • Hr -- DINT (hour) • Min -- DINT (minute) • Sec -- DINT (second) • uSec -- DINT (microsecond) 	T_LEQ_1_0-01_AOI.L5X
T_LES	DateTime < DateTime?	<p>This instruction compares two Date-and-Time-of-Day (DateTime) variables.</p> <p>If DT1 is less than (before) DT2, EnableOut and Out are set to true (1). Otherwise EnableOut and Out are cleared to false (0).</p> <p>This instruction can be used in ladder diagram, structured text, or function block Routines just like the LES instruction is used for integers and floating point numbers. However, because it has InOut Parameters (tag references to user-defined data types), it is not left justified on ladder rungs.</p> <p>On a False rung in LD, or in FBD with EnableIn cleared to 0, Out is cleared to 0.</p> <p>IMPORTANT: This instruction is dependent on the user-defined data type 'DateTime' (external to this Add-On Instruction definition).</p> <p>'DateTime' is defined as the following:</p> <ul style="list-style-type: none"> • Yr -- DINT (year) • Mo -- DINT (month) • Da -- DINT day) • Hr -- DINT (hour) • Min -- DINT (minute) • Sec -- DINT (second) • uSec -- DINT (microsecond) 	T_LES_1_0-01_AOI.L5X

Name	Short Description	Long Description	File Name
T_LIM	DateTime Limit Test	<p>This instruction compares a Date-and-Time-of-Day or amount of time with a High Limit (Date/Time or amount of time) and a Low Limit (Date/Time or amount of time).</p> <p>There are two cases:</p> <ul style="list-style-type: none"> • 'normal case' (Low Limit <= High Limit) • 'circular case' (Low Limit > High Limit) <p>In the normal case, EnableOut and Out are set if: Low Limit <= DateTime <= High Limit</p> <p>In the circular case, EnableOut and Out are set if: DateTime >= Low Limit OR DateTime <= High Limit (remember, High Limit < Low Limit)</p> <p>This instruction can be used in ladder diagram, structured text, or function block Routines just like the LIM instruction is used for integers and floating point numbers. However, because it has InOut Parameters (tag references to user-defined data types), it is not left justified on ladder rungs.</p> <p>On a False rung in LD, or in FBD with EnableIn cleared to 0, Out is cleared to 0.</p> <p>IMPORTANT: This instruction is dependent on the user-defined data type 'DateTime' (external to this Add-On Instruction definition).</p> <p>'DateTime' is defined as the following:</p> <ul style="list-style-type: none"> • Yr -- DINT (year) • Mo -- DINT (month) • Da -- DINT day) • Hr -- DINT (hour) • Min -- DINT (minute) • Sec -- DINT (second) • uSec -- DINT (microsecond) 	T_LIM_1_0-00_AOI.L5X
T_LtoT	LTIME to DateTime	<p>This instruction converts an LTIME (64-bit integer time stamp, for example, from an ALMD or ALMA instruction or the WallClockTime object) to a DateTime (year, month, day, hour, minute, second, microsecond as DINTs) in Coordinated Universal Time (UTC offset = 0).</p> <p>The input LTIME is the 64 bit (LINT) number of microseconds since DT#1970-01-01_00:00:00.000000 UTC.</p>	T_LtoT_1_0-01_AOI.L5X

Name	Short Description	Long Description	File Name
T_NEQ	DateTime <> DateTime?	<p>This instruction compares two Date-and-Time-of-Day (DateTime) variables.</p> <p>If DT1 is not equal to DT2, EnableOut and Out are set to true (1). Otherwise EnableOut and Out are cleared to false (0).</p> <p>This instruction can be used in ladder diagram, structured text, or function block Routines just like the NEQ instruction is used for integers and floating point numbers. However, because it has InOut Parameters (tag references to user-defined data types), it is not left justified on ladder rungs.</p> <p>On a False rung in LD, or in FBD with EnableIn cleared to 0, Out is cleared to 0.</p> <p>IMPORTANT: This instruction is dependent on the user-defined data type 'DateTime' (external to this Add-On Instruction definition).</p> <p>'DateTime' is defined as the following:</p> <ul style="list-style-type: none"> • Yr -- DINT (year) • Mo -- DINT (month) • Da -- DINT (day) • Hr -- DINT (hour) • Min -- DINT (minute) • Sec -- DINT (second) • uSec -- DINT (microsecond) 	T_NEQ_1_0-01_AOI.L5X
T_Now	Current DateTime	<p>Returns the current local date and time from the controller clock as a DateTime In/Out Parameter.</p> <p>This instruction depends on the (external) DateTime data type:</p> <ul style="list-style-type: none"> • Yr -- DINT (year) • Mo -- DINT (month) • Da -- DINT (day) • Hr -- DINT (hour) • Min -- DINT (minute) • Sec -- DINT (second) • uSec -- DINT (microsecond) 	T_Now_1_0-00_AOI.L5X
T_Scan	Time Since Previous Scan	Returns the time between the previous scan of the instance and the current scan of the same instance as a REAL number of Seconds.	T_Scan_1_0-01_AOI.L5X
T_SEL	DateTime Select	<p>This instruction uses an Input bit signal to select one of two Date/Time values.</p> <p>IMPORTANT: The selected Date/Time is only 'moved through' as-is; it is not 'normalized' to a valid Gregorian Date and Time: Inp_Sel is defaulted to 1 so this instruction can be used on a ladder diagram routine rung with the rung condition as the selector: Rung True selects DT1, Rung False selects DT0. The Inp_Sel is inverted when EnableIn is False (false rung). This inversion can be useful beyond this ladder diagram function.</p>	T_SEL_1_0-00_AOI.L5X

Name	Short Description	Long Description	File Name
T_SUB	DateTime:= DateTime - Time	<p>T_Sub: Subtract Date/Time minus time to get new Date/Time.</p> <p>This instruction subtracts a given amount of time from a Date/Time to arrive at a new Date/Time. The new Date/Time is 'normalized', that is, given as a valid (if possible) Gregorian Date and Time:</p> <ul style="list-style-type: none"> • 0 <= Microseconds < 1,000,000 • 0 <= Seconds < 60 (This instruction cannot add leap seconds) • 0 <= Minutes < 60 • 0 <= Hours < 24 • 1 <= Day <= 31 AND Date is a Valid Gregorian Date • 1 <= Month <= 12 	T_SUB_1_0-01_AOI.L5X
T_Sun	Sun Rise / Set / Az/El	<p>This instruction takes a given Date/Time, and for the date part, plus the configured latitude, longitude, and UTC offset, returns the Date/Time of local sunrise and local sunset (to the nearest minute, accurate to within about 2 minutes).</p> <p>Solar Azimuth (heading, clockwise in degrees from true north) and Elevation (degrees above horizon) are calculated and accurate to within about half a degree when the sun is above the horizon. Azimuth bearing is not necessarily accurate when elevation is more than a degree or two negative.</p> <p>The given date is assumed valid. If necessary, check by using T_Valid first.</p> <p>IMPORTANT: This instruction uses only the month and day to estimate the sunrise and sunset times to within a couple minutes. It does not deal with detailed astronomical calculations that are based on planetary models. It is based on the current Gregorian calendar and does not deal with Julian dates for dates before 1582.</p> <p>For algorithms, see: http://www.srrb.noaa.gov/highlights/sunrise/solareqns.PDF</p> <p>To get your latitude and longitude, see: http://www.batchgeocode.com/lookup/</p>	T_Sun_1_0-00_AOI.L5X

Name	Short Description	Long Description	File Name
T_Sync	Synchronize Controller Clock	<p>This object synchronizes the controller real-time clock with an NTP Time Server with excellent reliability (computer responsible for network time, or a standard time server, like time.windows.com)</p> <p>IMPORTANT: This instruction is not a full NTP precision exchange. It is simply a quick 'get' of the NTP server time and the instruction applies it to the controller clock.</p> <p>It supports the following features:</p> <ul style="list-style-type: none"> • Ability to sync controller clock to server on Maintenance command • Ability to sync on a periodic (default = daily) basis • Ability to retry on a periodic (default = hourly) basis on a failure to retrieve date/time from server • Ability to sync on controller powerup or PROGRAM to RUN transition. • Ability via configuration to allow or disallow each of the previous methods • Reads time from NTP server and displays time that is received as Values. • Updates the controller clock to time received (if allowed by configuration) • Calculates clock drift (difference between previous and new controller time) and displays as Values. <p>Cfg_PollT: The Poll Time (default = 1440 min = 1 day) is the number of minutes between polls of the NTP server for excellent time reliability after a successful get of the time.</p> <p>Cfg_RetryT: The Retry Time (default = 60 min = 1 hour) is the number of minutes between polls of the NTP server for excellent time reliability after a failure to get the time.</p> <p>Cfg_ENSlotNumber: Enter the chassis slot number of the EtherNet/IP module (for example, 1756-EN2T) that can communicate with the time server. This module must support 'socket services'. For more information, see the EtherNet/IP Socket Interface Application Technique, publication ENET-AT002.</p> <p>TIP: For CompactLogix™ controllers with built-in Ethernet interface (for example, the 1769-L36ERM), use a value of 1.</p> <p>IMPORTANT: The Cfg_ENSlotNumber value is used to build the Path for the MSG instructions that are used in T_Sync. If you change Cfg_ENSlotNumber while the controller is running, you can cycle the controller to PROG and back to RUN for the Path change to take effect.</p> <p>Cfg_Host (a Local STRING tag): Name or IP address of the time server.</p> <p>IMPORTANT: If you use the name of the host, be sure the DNS (domain name service) server addresses are configured for your Ethernet module so the name can be resolved to an IP address. The default value of 'time.nist.gov' for users in the United States provides an automatic redirect to an available time server with excellent reliability. For a local domain, the domain controller often provides time service for members of the domain; if it has a fixed IP address, you can enter it in the common dotted-decimal format, for example '192.168.1.1'.</p> <p>Cfg_Port (a Local STRING tag): Number of the IP Port for the NTP time service. This string must be in the form: '?port=nnn' where nnn is the port number in decimal. The default NTP port number is 123, so the default value of '?port=123' works for most cases.</p> <p>Cfg_AllowClockUpdate:</p> <p>1 = Allow Add-On Instruction to update controller clock. 0 = Get the time (UTC) and display it.</p> <p>Cfg_AllowMCmdSync:</p> <p>0 = No manual sync request allowed.</p>	<p>T_Sync_1_0-05_AOI.L5X</p> <p>T_Sync_1_0-05_RUNG.L5X</p>

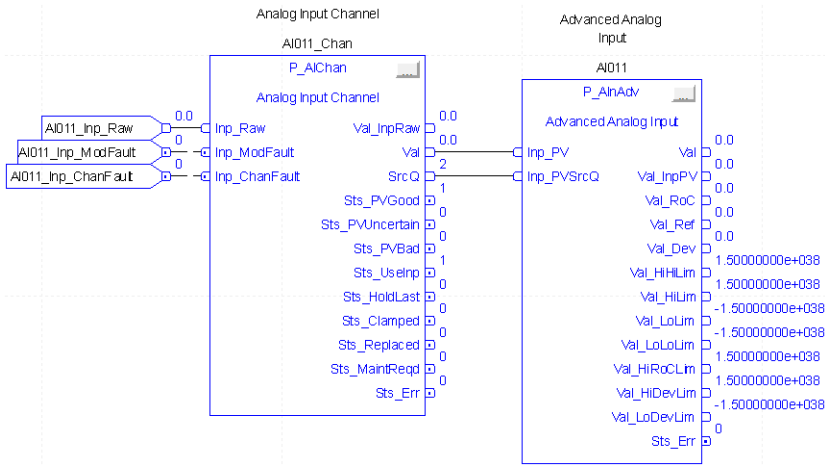
Name	Short Description	Long Description	File Name
T_TtoISO_WkDate	Date to ISO Week Date	<p>This object converts a Date in common form (Year, Month, Day) to an ISO-8601 Week Date (like 2014-W04-2, meaning Tuesday in the fourth week of Week-Numbered Year 2014) for 2014-01-21.</p> <p>This object calculates the ISO Year, ISO Week, and ISO Day (day of the week). The ISO Day is specified as 1=Monday ...7=Sunday. This object also determines the number of weeks (52 or 53) in the calculated ISO Year.</p> <p>The Date to convert is given in the Year, Month, and Day of the Ref_DT reference tag, of type DateTime (Year, Month, Day, Hour, Minute, Second, Microsecond).</p> <p>The Week-Numbered Year does not necessarily start on January 1. It can start from December 29 through January 4. The first week of an ISO Week-Numbered Year is the week, beginning with Monday and ending on Sunday, which contains the first THURSDAY of the calendar year. See Wikipedia, 'ISO Week Date', for more information.</p>	T_TtoISO_WkDate_1_0-00_AOI.L5X
T_TtoL	DateTime to LTIME	<p>This instruction converts a DateTime (year, month, day, hour, minute, second, microsecond as DINTs) in Coordinated Universal Time (UTC offset = 0) to an LTIME (64-bit integer time stamp, for example, from an ALMD or ALMA instruction or the WallClockTime object).</p> <p>The output LTIME is the 64-bit (LINT) number of microseconds since DT#1970-01-01_00:00:00.0000 UTC.</p>	T_TtoL_1_0-00_AOI.L5X
T_TtoS	Date/Time to String	<p>This instruction takes the given date and formats it as a human-readable STRING.</p> <p>For example, for the Date/Time: 2008 12 31 23 59 59 999999 the return STRING is (based on configuration): Wednesday, December 31, 2008 11:59:59.999999 p.m.</p> <p>Options are provided for:</p> <ul style="list-style-type: none"> • 24- or 12-hour time format (with a.m. or p.m. indicator on the 12-hour format) • Displaying or not displaying microseconds • Displaying or not displaying seconds • Displaying or not displaying the Day of the Week • Displaying day first (31 July) or month first (July 31) • Displaying date in an ISO-format (YYYY-MM-DD) <p>This instruction checks for a valid (Gregorian) date and time and returns 'Invalid Date' and/or 'Invalid Time' as appropriate in the output STRING. The following are valid dates/times:</p> <ul style="list-style-type: none"> • 0 <= Microseconds < 1,000,000 • 0 <= Seconds < 60 • 0 <= Minutes < 60 • 0 <= Hours < 24 • 1 <= Days <= 31 and a valid Gregorian Date (Feb = 28 or 29 days) • 1 <= Month <= 12 • Year in the range +/- 5879600 <p>The names of the days of the week and the months of the year, and the AM and PM indicator text can be changed by using the Local Tags Monitor for the instance.</p>	T_TtoS_1_0-00_AOI.L5X

Name	Short Description	Long Description	File Name
T_Valid	Is DateTime Valid?	<p>This instruction tests the given DateTime variable and verifies that it is a valid calendar date and clock time, as follows:</p> <ul style="list-style-type: none">• 0 ≤ Microseconds < 1,000,000• 0 ≤ Seconds < 60 (This instruction cannot check leap seconds)• 0 ≤ Minutes < 60• 0 ≤ Hours < 24• 1 ≤ Day ≤ 31 and Day is Valid for Gregorian Date (28, 29, 30, or 31 days in month)• 1 ≤ Month ≤ 12• Year is within the range of dates that this instruction can calculate a Gregorian day number (about +/- 5.8 million years) <p>IMPORTANT: This instruction does not switch to Julian dates for dates before 1582 (or 1753). This instruction assumes that the Gregorian Calendar extends 'indefinitely' (at least 5.8 million years) either side of 'zero'. It does handle the Gregorian 4-, 100-, 400-year rules, so Feb. 29, 2000 is a valid date, but Feb. 29, 2100 is not.</p>	T_Valid_1_0-00_AOI.L5X

Process Strategies

Process Strategies

Process strategies incorporate the Rockwell Automation Library of Process Objects. To meet control system needs, process strategies provide pre-connected functionality. These sets of connected Process Library objects help reduce implementation time and improve control objectives for process devices.



The strategies can be imported as Function Block routines and Ladder Logic rung imports. For more information, see the PlantPAx System Application Configuration User Manual, publication [PROCES-UM003](#).

The following table provides a summary of the available Process Strategies:

Table 25 - Process Strategies

Object Category	Description	Additional Process Library Objects Used	Process Strategy
I/O Processing	Discrete Input Object (P_DIn)	None	(RA-LIB)PS_DIn_4_10-00_ROUTINE.L5X
	Discrete Output (P_DOut)	• P_Intlk	(RA-LIB)PS_DOut_4_10-00_ROUTINE.L5X
	Discrete Output (P_DOut) without Interlocks	None	(RA-LIB)PS_DOut_noIntlk_4_10-00_ROUTINE.L5X
	Basic Analog Input (P_AIn)	None	(RA-LIB)PS_AIn_4_10-00_ROUTINE.L5X
	Basic Analog Input (P_AIn) with Analog Input Channel Diagnostics	• P_AIChan	(RA-LIB)PS_AIn_Chان_4_10-00_ROUTINE.L5X
	Advanced Analog Input (P_AInAdv)	None	(RA-LIB)PS_AInAdv_4_10-00_ROUTINE.L5X
	Advanced Analog Input (P_AInAdv) with Analog Input Channel Diagnostics	• P_AIChan	(RA-LIB)PS_AInAdv_Chان_4_10-00_ROUTINE.L5X
	Dual Sensor Analog Input (P_AInDual)	None	(RA-LIB)PS_AInDual_4_10-00_ROUTINE.L5X
	Dual Sensor Analog Input (P_AInDual) with Analog Input Channel Diagnostics	• P_AIChan (2)	(RA-LIB)PS_AInDual_Chان_4_10-00_ROUTINE.L5X
	Multiple Analog Input (P_AInMulti)	None	(RA-LIB)PS_AInMulti_4_10-00_ROUTINE.L5X
	Analog Output (P_AOut)	• P_Intlk	(RA-LIB)PS_AOut_4_10-00_ROUTINE.L5X
	Analog Output (P_AOut) without Interlocks	None	(RA-LIB)PS_AOut_noIntlk_4_10-00_ROUTINE.L5X
	Discrete Input Object Advanced (P_DInAdv)		(RA-LIB)PS_DInAdv_4_10-00_ROUTINE.L5X
	Discrete Output (P_DOut)	• P_IntlkAdv	(RA-LIB)PS_DOut_IntlkAdv_4_10-00_ROUTINE.L5X
	Discrete Output (P_DOut) without Interlocks		(RA-LIB)PS_DOut_noIntlk_4_10-00_ROUTINE.L5X
	Analog Output (P_AOut)		(RA-LIB)PS_AOut_IntlkAdv_4_10-00_ROUTINE.L5X

Table 25 - Process Strategies

Object Category	Description	Additional Process Library Objects Used	Process Strategy
Regulatory Control	PID Enhanced (P_PIDE)	• P_IntlkAdv	(RA-LIB)PS_PID_IntlkAdv_4_10-00_ROUTINE.L5X
	PID Enhanced (P_PIDE) with Basic Analog Input	• P_AIn • P_IntlkAdv	(RA-LIB)PS_PID_AIn_IntlkAdv_4_10-00_ROUTINE.L5X
	PID Enhanced (P_PIDE) with Basic Analog Input and Channel Diagnostics	• P_AIn • P_AIChan • P_IntlkAdv	(RA-LIB)PS_PID_AIn_Chan_IntlkAdv_4_10-00_ROUTINE.L5X
	PID Enhanced (P_PIDE) with Advanced Analog Input	• P_AInAdv • P_IntlkAdv	(RA-LIB)PS_PID_AInAdv_IntlkAdv_4_10-00_ROUTINE.L5X
	PID Enhanced (P_PIDE) with Advanced Analog Input and Channel Diagnostics	• P_AInAdv • P_AIChan • P_IntlkAdv	(RA-LIB)PS_PID_AInAdv_Chan_IntlkAdv_4_10-00_ROUTINE.L5X
	PID Enhanced (P_PIDE) Cascade with Basic Analog Input and Channel Diagnostics: Primary (outer) Loop	• P_AIn • P_AIChan • P_IntlkAdv	(RA-LIB)PS_PID_Casc_IntlkAdv_AIn_Pri_4_10-00_ROUTINE
	PID Enhanced (P_PIDE) Cascade with Basic Analog Input and Channel Diagnostics: Secondary (inner) Loop	• P_AIn • P_AIChan • P_IntlkAdv	(RA-LIB)PS_PID_Casc_IntlkAdv_AIn_Sec_4_10-00_ROUTINE
	PID Enhanced (P_PIDE) Cascade with Advanced Analog Input and Channel Diagnostics: Primary (outer) Loop	• P_AInAdv • P_AIChan • P_IntlkAdv	(RA-LIB)PS_PID_Casc_IntlkAdv_AInAdv_Pri_4_10-00_ROUTINE
	PID Enhanced (P_PIDE) Cascade with Advanced Analog Input and Channel Diagnostics: Secondary (inner) Loop	• P_AInAdv • P_AIChan • P_IntlkAdv	(RA-LIB)PS_PID_Casc_IntlkAdv_AInAdv_Sec_4_10-00_ROUTINE
	PID Enhanced (P_PIDE) Cascade with Basic Analog Input, PowerFlex 755 Drive: Primary (outer) Loop	• P_AIn • P_PF755	(RA-LIB)PS_PID_PF755_IntlkAdv_AIn_Pri_4_10-00_ROUTINE
	PID Enhanced (P_PIDE) Cascade with Basic Analog Input, PowerFlex 755 Drive: Secondary (inner) Loop with PowerFlex755 Drive as Final Control Element	• P_PF755 • P_IntlkAdv • P_Perm • P_RunTime	(RA-LIB)PS_PID_PF755_IntlkAdv_AIn_Sec_4_10-00_ROUTINE
	Ramp / Soak (RMPS)	None	(RA-LIB)PS_RMPS_4_10-00_ROUTINE.L5X
	Analog Fanout (P_Fanout)	None	(RA-LIB)PS_Fanout_4_10-00_ROUTINE.L5X
	High or Low Selector (P_HiLoSel)	None	(RA-LIB)PS_HiLoSel_4_10-00_ROUTINE.L5X
Model-Based Control	Internal Model Control (IMC)	None	(RA-LIB)PS_IMC_4_10-00_ROUTINE.L5X
	Coordinated Control (CC)	None	(RA-LIB)PS_CC_4_10-00_ROUTINE.L5X
	Modular Multivariable Control (MMC)	None	(RA-LIB)PS_MMC_4_10-00_ROUTINE.L5X

Table 25 - Process Strategies

Object Category	Description	Additional Process Library Objects Used	Process Strategy
Procedural Control	Dosing (P_Dose) with Flowmeter using Basic Analog Input and Channel Diagnostics	<ul style="list-style-type: none"> • P_AIn • P_AIChan 	(RA-LIB)PS_DoseFM_4_10-00_ROUTINE.L5X
	Dosing (P_Dose) with Weigh Scale	None	(RA-LIB)PS_DoseWS_4_10-00_ROUTINE.L5X
	Sequencer Object (P_Seq) with Prompt	<ul style="list-style-type: none"> • P_Prompt • P_Intlk • P_Perm 	(RA-LIB)PS_Seq_4_10-00_ROUTINE.L5X
	Sequencer Object (P_Seq) without Prompt	<ul style="list-style-type: none"> • P_Intlk • P_Perm 	(RA-LIB)PS_Seq_noPrompt_4_10-00_ROUTINE.L5X
	Sequencer Object (P_Seq) with Prompt	<ul style="list-style-type: none"> • P_Prompt • P_IntlkAdv • P_Perm 	(RA-LIB)PS_Seq_IntlkAdv_4_10-00_ROUTINE.L5X
	Sequencer Object (P_Seq) without Prompt	<ul style="list-style-type: none"> • P_IntlkAdv • P_Perm 	(RA-LIB)PS_Seq_noPrompt_IntlkAdv_4_10-00_ROUTINE.L5X
	Totalizer	None	(RA-LIB)PS_Tot_4_10-00_ROUTINE.L5X
Valves	Control Valve	<ul style="list-style-type: none"> • P_Intlk 	(RA-LIB)PS_ValveC_4_10-00_ROUTINE.L5X
	Control Valve	<ul style="list-style-type: none"> • P_IntlkAdv 	(RA-LIB)PS_ValveC_IntlkAdv_4_10-00_ROUTINE.L5X
	Hand-Operated Valve	<ul style="list-style-type: none"> • P_Intlk • P_ValveStats 	(RA-LIB)PS_ValveHO_4_10-00_ROUTINE.L5X
	Hand-Operated Valve	<ul style="list-style-type: none"> • P_IntlkAdv • P_ValveStats 	(RA-LIB)PS_ValveHO_IntlkAdv_4_10-00_ROUTINE.L5X
	Motor-Operated Valve	<ul style="list-style-type: none"> • P_Perm (2) • P_ValveStats 	(RA-LIB)PS_ValveMO_4_10-00_ROUTINE.L5X
	Mix-Proof Valve	<ul style="list-style-type: none"> • P_Intlk • P_Perm • P_ValveStats 	(RA-LIB)PS_ValveMP_4_10-00_ROUTINE.L5X
	Mix-Proof Valve	<ul style="list-style-type: none"> • P_Intlk • P_Perm • P_ValveStats 	(RA-LIB)PS_ValveMP_IntlkAdv_4_10-00_ROUTINE.L5X
	Solenoid-Operated Valve	<ul style="list-style-type: none"> • P_Intlk • P_Perm • P_ValveStats 	(RA-LIB)PS_ValveSO_4_10-00_ROUTINE.L5X
	Solenoid-Operated Valve	<ul style="list-style-type: none"> • P_IntlkAdv • P_Perm • P_ValveStats 	(RA-LIB)PS_ValveSO_IntlkAdv_4_10-00_ROUTINE.L5X

Table 25 - Process Strategies

Object Category	Description	Additional Process Library Objects Used	Process Strategy
Single-Speed Motors	Single-speed Motor (P_Motor)	<ul style="list-style-type: none"> • P_Intlk • P_Perm • P_Runtime • P_ResInh 	(RA-LIB)PS_Motor_4_10-00_ROUTINE.L5X
	Single-speed Motor (P_Motor)	<ul style="list-style-type: none"> • P_IntlkAdv • P_Perm • P_Runtime • P_ResInh 	(RA-LIB)PS_Motor_IntlkAdv_4_10-00_ROUTINE.L5X
	Single-speed Motor (P_Motor) with E300 Overload Relay	<ul style="list-style-type: none"> • P_E300Ovld • P_Intlk • P_Perm • P_Runtime • P_ResInh 	(RA-LIB)PS_Motor_E300Ovld_4_10-00_ROUTINE.L5X
	Single-speed Motor (P_Motor) with E300 Overload Relay	<ul style="list-style-type: none"> • P_E300Ovld • P_IntlkAdv • P_Perm • P_Runtime • P_ResInh 	(RA-LIB)PS_Motor_E300Ovld_IntlkAdv_4_10-00_ROUTINE.L5X
	Single-speed Motor (P_Motor) with E3 / E3Plus Overload Relay	<ul style="list-style-type: none"> • P_E3Ovld • P_Intlk • P_Perm • P_Runtime • P_ResInh 	(RA-LIB)PS_Motor_E3Ovld_4_10-00_ROUTINE.L5X
	Single-speed Motor (P_Motor) with E3 / E3Plus Overload Relay	<ul style="list-style-type: none"> • P_E3Ovld • P_IntlkAdv • P_Perm • P_Runtime • P_ResInh 	(RA-LIB)PS_Motor_E3Ovld_IntlkAdv_4_10-00_ROUTINE.L5X
	Single-speed Motor (P_Motor) with E1Plus	<ul style="list-style-type: none"> • P_E1PlusE • P_Intlk • P_Perm • P_Runtime • P_ResInh 	(RA-LIB)PS_Motor_E1PlusE_4_10-00_ROUTINE.L5X
	Single-speed Motor (P_Motor) with E1Plus	<ul style="list-style-type: none"> • P_E1PlusE • P_IntlkAdv • P_Perm • P_Runtime • P_ResInh 	(RA-LIB)PS_Motor_E1PlusE_IntlkAdv_4_10-00_ROUTINE.L5X

Table 25 - Process Strategies

Object Category	Description	Additional Process Library Objects Used	Process Strategy
Two-Speed Motors	Two-speed Motor (P_Motor2Spd)	<ul style="list-style-type: none"> • P_Intlk • P_Perm (2) • P_Runtime • P_ResInh 	(RA-LIB)PS_Motor2Spd_4_10-00_ROUTINE.L5X
	Two-speed Motor (P_Motor2Spd)	<ul style="list-style-type: none"> • P_IntlkAdv • P_Perm (2) • P_Runtime • P_ResInh 	(RA-LIB)PS_Motor2Spd_IntlkAdv_4_10-00_ROUTINE.L5X
	Two-speed Motor (P_Motor2Spd) with E300 Overload Relay	<ul style="list-style-type: none"> • P_E300Ovld • P_Intlk • P_Perm (2) • P_Runtime • P_ResInh 	(RA-LIB)PS_Motor2Spd_E300Ovl_4_10-00_ROUTINE.L5X
	Two-speed Motor (P_Motor2Spd) with E300 Overload Relay	<ul style="list-style-type: none"> • P_E300Ovld • P_IntlkAdv • P_Perm (2) • P_Runtime • P_ResInh 	(RA-LIB)PS_Motor2Spd_E300Ovld_IntlkAdv_4_10-00_ROUTINE.L5X
	Two-speed Motor (P_Motor2Spd) with E3 / E3Plus Overload Relay	<ul style="list-style-type: none"> • P_E3Ovld • P_Intlk • P_Perm (2) • P_Runtime • P_ResInh 	(RA-LIB)PS_Motor2Spd_E3Ovl_4_10-00_ROUTINE.L5X
	Two-speed Motor (P_Motor2Spd) with E3 / E3Plus Overload Relay	<ul style="list-style-type: none"> • P_E3Ovld • P_IntlkAdv • P_Perm (2) • P_Runtime • P_ResInh 	(RA-LIB)PS_Motor2Spd_E3Ovld_IntlkAdv_4_10-00_ROUTINE.L5X

Table 25 - Process Strategies

Object Category	Description	Additional Process Library Objects Used	Process Strategy
Motor Monitors (Hand-Operated Motors)	Hand-Operated Motor (P_MotorHO)	<ul style="list-style-type: none"> • P_Intlk • P_Perm • P_Runtime • P_ResInh 	(RA-LIB)PS_MotorHO_4_10-00_ROUTINE.L5X
	Hand-Operated Motor (P_MotorHO)	<ul style="list-style-type: none"> • P_IntlkAdv • P_Perm • P_Runtime • P_ResInh 	(RA-LIB)PS_MotorHO_IntlkAdv_4_10-00_ROUTINE.L5X
	Hand-Operated Motor (P_MotorHO) with E300 Overload Relay	<ul style="list-style-type: none"> • P_E300Ovld • P_Intlk • P_Perm • P_Runtime • P_ResInh 	(RA-LIB)PS_MotorHO_E300Ovl_4_10-00_ROUTINE.L5X
	Hand-Operated Motor (P_MotorHO) with E300 Overload Relay	<ul style="list-style-type: none"> • P_E300Ovld • P_IntlkAdv • P_Perm • P_Runtime • P_ResInh 	(RA-LIB)PS_MotorHO_E300Ovld_IntlkAdv_4_10-00_ROUTINE.L5X
	Hand-Operated Motor (P_MotorHO) with E3 / E3Plus Overload Relay	<ul style="list-style-type: none"> • P_E3Ovld • P_Intlk • P_Perm • P_Runtime • P_ResInh 	(RA-LIB)PS_MotorHO_E3Ovl_4_10-00_ROUTINE.L5X
	Hand-Operated Motor (P_MotorHO) with E3 / E3Plus Overload Relay	<ul style="list-style-type: none"> • P_E3Ovld • P_IntlkAdv • P_Perm • P_Runtime • P_ResInh 	(RA-LIB)PS_MotorHO_E3Ovld_IntlkAdv_4_10-00_ROUTINE.L5X
	Single-speed Motor (P_MotorHO) with E1Plus	<ul style="list-style-type: none"> • P_E1PlusE • P_Intlk • P_Perm • P_Runtime • P_ResInh 	(RA-LIB)PS_MotorHO_E1PlusE_4_10-00_ROUTINE.L5X
	Single-speed Motor (P_MotorHO) with E1Plus	<ul style="list-style-type: none"> • P_E1PlusE • P_IntlkAdv • P_Perm • P_Runtime • P_ResInh 	(RA-LIB)PS_MotorHO_E1PlusE_IntlkAdv_4_10-00_ROUTINE.L5X

Table 25 - Process Strategies

Object Category	Description	Additional Process Library Objects Used	Process Strategy
Reversing Motors	Reversing Motor (P_MotorRev)	<ul style="list-style-type: none"> • P_Intlk • P_Perm (2) • P_Runtime • P_ResInh 	(RA-LIB)PS_MotorRev_4_10-00_ROUTINE.L5X
	Reversing Motor (P_MotorRev)	<ul style="list-style-type: none"> • P_IntlkAdv • P_Perm (2) • P_Runtime • P_ResInh 	(RA-LIB)PS_MotorRev_IntlkAdv_4_10-00_ROUTINE.L5X
	Two-speed Motor (P_Motor2Spd) with E300 Overload Relay	<ul style="list-style-type: none"> • P_E300Ovld • P_Intlk • P_Perm (2) • P_Runtime • P_ResInh 	(RA-LIB)PS_MotorRev_E300Ovl_4_10-00_ROUTINE.L5X
	Two-speed Motor (P_Motor2Spd) with E300 Overload Relay	<ul style="list-style-type: none"> • P_E300Ovld • P_IntlkAdv • P_Perm (2) • P_Runtime • P_ResInh 	(RA-LIB)PS_MotorRev_E300OVld_IntlkAdv_4_10-00_ROUTINE.L5X
	Two-speed Motor (P_Motor2Spd) with E3 / E3Plus Overload Relay	<ul style="list-style-type: none"> • P_E3Ovld • P_Intlk • P_Perm (2) • P_Runtime • P_ResInh 	(RA-LIB)PS_MotorRev_E3Ovl_4_10-00_ROUTINE.L5X
	Two-speed Motor (P_Motor2Spd) with E3 / E3Plus Overload Relay	<ul style="list-style-type: none"> • P_E3Ovld • P_IntlkAdv • P_Perm (2) • P_Runtime • P_ResInh 	(RA-LIB)PS_MotorRev_E3OVld_IntlkAdv_4_10-00_ROUTINE.L5X

Table 25 - Process Strategies

Object Category	Description	Additional Process Library Objects Used	Process Strategy
Drives	PowerFlex 525 Variable-Frequency Drive	<ul style="list-style-type: none"> • P_Intlk • P_Perm • P_Runtime 	(RA-LIB)PS_PF52x_4_10-00_ROUTINE.L5X
	PowerFlex 525 Variable-Frequency Drive	<ul style="list-style-type: none"> • P_IntlkAdv • P_Perm • P_Runtime 	(RA-LIB)PS_PF52x_IntlkAdv_4_10-00_ROUTINE.L5X
	PowerFlex 753 Variable-Frequency Drive	<ul style="list-style-type: none"> • P_Intlk • P_Perm • P_Runtime 	(RA-LIB)PS_PF753_4_10-00_ROUTINE.L5X
	PowerFlex 753 Variable-Frequency Drive	<ul style="list-style-type: none"> • P_IntlkAdv • P_Perm • P_Runtime 	(RA-LIB)PS_PF753_IntlkAdv_4_10-00_ROUTINE.L5X
	PowerFlex 755 Variable-Frequency Drive	<ul style="list-style-type: none"> • P_Intlk • P_Perm • P_Runtime 	(RA-LIB)PS_PF755_4_10-00_ROUTINE.L5X
	PowerFlex 755 Variable-Frequency Drive	<ul style="list-style-type: none"> • P_IntlkAdv • P_Perm • P_Runtime 	(RA-LIB)PS_PF755_4_10-00_ROUTINE.L5X
	PowerFlex 6000 Medium Voltage Variable-Frequency Drive	<ul style="list-style-type: none"> • P_Intlk • P_Perm • P_Runtime 	(RA-LIB)PS_PF6000_4_10-00_ROUTINE.L5X
	PowerFlex 6000 Medium Voltage Variable-Frequency Drive	<ul style="list-style-type: none"> • P_IntlkAdv • P_Perm • P_Runtime 	(RA-LIB)PS_PF6000_IntlkAdv_4_10-00_ROUTINE.L5X
	PowerFlex 7000 Medium Voltage Variable-Frequency Drive	<ul style="list-style-type: none"> • P_Intlk • P_Perm • P_Runtime 	(RA-LIB)PS_PF7000_4_10-00_ROUTINE.L5X
	PowerFlex 7000 Medium Voltage Variable-Frequency Drive	<ul style="list-style-type: none"> • P_IntlkAdv • P_Perm • P_Runtime 	(RA-LIB)PS_PF7000_IntlkAdv_4_10-00_ROUTINE.L5X
	Generic Variable-Frequency Drive (via discrete/analog signals)	<ul style="list-style-type: none"> • P_Intlk • P_Perm • P_Runtime 	(RA-LIB)PS_VSD_4_10-00_ROUTINE.L5X
	Generic Variable-Frequency Drive (via discrete/analog signals)	<ul style="list-style-type: none"> • P_IntlkAdv • P_Perm • P_Runtime 	(RA-LIB)PS_VSD_IntlkAdv_4_10-00_ROUTINE.L5X
Smart Motor Controllers (Soft Starters)	SMC-50 Smart Motor Controller (P_SMC50)	<ul style="list-style-type: none"> • P_Intlk • P_Perm • P_Runtime 	(RA-LIB)PS_SMC50_4_10-00_ROUTINE.L5X
	SMC-50 Smart Motor Controller (P_SMC50)	<ul style="list-style-type: none"> • P_IntlkAdv • P_Perm • P_Runtime 	(RA-LIB)PS_SMC50_IntlkAdv_4_10-00_ROUTINE.L5X
	SMCFlex Smart Motor Controller (P_SMCFlex)	<ul style="list-style-type: none"> • P_Intlk • P_Perm • P_Runtime 	(RA-LIB)PS_SMCFlex_4_10-00_ROUTINE.L5X
	SMCFlex Smart Motor Controller (P_SMCFlex)	<ul style="list-style-type: none"> • P_IntlkAdv • P_Perm • P_Runtime 	(RA-LIB)PS_SMCFlex_IntlkAdv_4_10-00_ROUTINE.L5X

Table 25 - Process Strategies

Object Category	Description	Additional Process Library Objects Used	Process Strategy
Low Voltage Motor Control Centers	Full-Voltage Non-Reversing Motor (P_Motor) with E300 Overload Relay, no run feedback (Operating Mode 3)	<ul style="list-style-type: none"> • P_E300Ovld • P_Intlk • P_Perm • P_ResInh • P_Runtime 	(RA-LIB)PS_LVMCC_FVNR_E300Ovld_OpMode3_4_10-00_ROUTINE.L5X
	Full-Voltage Non-Reversing Motor (P_Motor) with E300 Overload Relay, no run feedback (Operating Mode 3)	<ul style="list-style-type: none"> • P_E300Ovld • P_IntlkAdv • P_Perm • P_ResInh • P_Runtime 	(RA-LIB)PS_LVMCC_FVNR_E300Ovld_OpMode3_IntlkAdv_4_10-00_ROUTINE.L5X
	Full-Voltage Non-Reversing Motor (P_Motor) with E300 Overload Relay, run feedback wired (Operating Mode 11)	<ul style="list-style-type: none"> • P_E300Ovld • P_Intlk • P_Perm • P_ResInh • P_Runtime 	(RA-LIB)PS_LVMCC_FVNR_E300Ovld_OpMode11_4_10-00_ROUTINE.L5X
	Full-Voltage Non-Reversing Motor (P_Motor) with E300 Overload Relay, run feedback wired (Operating Mode 11)	<ul style="list-style-type: none"> • P_E300Ovld • P_IntlkAdv • P_Perm • P_ResInh • P_Runtime 	(RA-LIB)PS_LVMCC_FVNR_E300Ovld_OpMode11_IntlkAdv_4_10-00_ROUTINE.L5X
	Full-Voltage Non-Reversing Motor (P_Motor) with E300 Overload Relay, hardwired Hand/Off/Auto (Operating Mode 2)	<ul style="list-style-type: none"> • P_E300Ovld • P_Intlk • P_Perm • P_ResInh • P_Runtime 	(RA-LIB)PS_LVMCC_FVNR_HOA_E300Ovld_OpMode2_4_10-00_ROUTINE.L5X
	Full-Voltage Non-Reversing Motor (P_Motor) with E300 Overload Relay, hardwired Hand/Off/Auto (Operating Mode 2)	<ul style="list-style-type: none"> • P_E300Ovld • P_IntlkAdv • P_Perm • P_ResInh • P_Runtime 	(RA-LIB)PS_LVMCC_FVNR_HOA_E300Ovld_OpMode2_IntlkAdv_4_10-00_ROUTINE.L5X
	Full-Voltage Reversing Motor (P_MotorRev) with E300 Overload Relay, no run feedback (Operating Mode 5)	<ul style="list-style-type: none"> • P_E300Ovld • P_Intlk • P_Perm (2) • P_ResInh • P_Runtime 	(RA-LIB)PS_LVMCC_FVR_E300Ovld_OpMode5_4_10-00_ROUTINE.L5X
	Full-Voltage Reversing Motor (P_MotorRev) with E300 Overload Relay, no run feedback (Operating Mode 5)	<ul style="list-style-type: none"> • P_E300Ovld • P_IntlkAdv • P_Perm (2) • P_ResInh • P_Runtime 	(RA-LIB)PS_LVMCC_FVR_E300Ovld_OpMode5_IntlkAdv_4_10-00_ROUTINE.L5X
	Full-Voltage Reversing Motor (P_MotorRev) with E300 Overload Relay, run feedback wired (Operating Mode 13)	<ul style="list-style-type: none"> • P_E300Ovld • P_Intlk • P_Perm (2) • P_ResInh • P_Runtime 	(RA-LIB)PS_LVMCC_FVR_E300Ovld_OpMode13_4_10-00_ROUTINE.L5X
	Full-Voltage Reversing Motor (P_MotorRev) with E300 Overload Relay, run feedback wired (Operating Mode 13)	<ul style="list-style-type: none"> • P_E300Ovld • P_IntlkAdv • P_Perm (2) • P_ResInh • P_Runtime 	(RA-LIB)PS_LVMCC_FVR_E300Ovld_OpMode13_IntlkAdv_MainRoutine_4_10-00_ROUTINE.L5X
	PowerFlex 525 Variable-Frequency Drive (P_PF52x) with hardwired Hand/Off/Auto switch	<ul style="list-style-type: none"> • P_Intlk • P_Perm • P_Runtime 	(RA-LIB)PS_LVMCC_PF525_with_HOA_4_10-00_ROUTINE.L5X
	PowerFlex 525 Variable-Frequency Drive (P_PF52x) with hardwired Hand/Off/Auto switch	<ul style="list-style-type: none"> • P_IntlkAdv • P_Perm • P_Runtime 	(RA-LIB)PS_LVMCC_PF525_with_HOA_IntlkAdv_4_10-00_ROUTINE.L5X

Table 25 - Process Strategies

Object Category	Description	Additional Process Library Objects Used	Process Strategy
Low Voltage Motor Control Centers	PowerFlex 753 Variable-Frequency Drive (P_PF753) with hardwired Hand/Off/Auto switch	<ul style="list-style-type: none"> • P_Intlk • P_Perm • P_Runtime 	(RA-LIB)PS_LVMCC_PF753_with_HOA_4_10-00_ROUTINE.L5X
	PowerFlex 753 Variable-Frequency Drive (P_PF753) with hardwired Hand/Off/Auto switch	<ul style="list-style-type: none"> • P_IntlkAdv • P_Perm • P_Runtime 	(RA-LIB)PS_LVMCC_PF753_with_HOA_IntlkAdv_4_10-00_ROUTINE.L5X
	PowerFlex 755 Variable-Frequency Drive (P_PF755) with hardwired Hand/Off/Auto switch	<ul style="list-style-type: none"> • P_Intlk • P_Perm • P_Runtime 	(RA-LIB)PS_LVMCC_PF755_with_HOA_4_10-00_ROUTINE.L5X
	PowerFlex 755 Variable-Frequency Drive (P_PF755) with hardwired Hand/Off/Auto switch	<ul style="list-style-type: none"> • P_IntlkAdv • P_Perm • P_Runtime 	(RA-LIB)PS_LVMCC_PF755_with_HOA_IntlkAdv_4_10-00_ROUTINE.L5X
Other	Discrete 2-, 3-, or 4-State Device (P_D4SD)	<ul style="list-style-type: none"> • P_Intlk • P_Perm 	(RA-LIB)PS_D4SD_4_10-00_ROUTINE.L5X
	Discrete 2-, 3-, or 4-State Device (P_D4SD)	<ul style="list-style-type: none"> • P_IntlkAdv • P_Perm 	(RA-LIB)PS_D4SD_IntlkAdv_4_10-00_ROUTINE.L5X
	Configurable Boolean Logic (8-gates) (P_Logic)	None	(RA-LIB)PS_Logic_4_10-00_ROUTINE.L5X
	I/O Module Connection Status (RUNG import)	None	(RA-LIB)PS_ModuleSts_4_10-00_RUNG.L5X
	Discrete n-Position (up to 8) Device (P_nPos)	<ul style="list-style-type: none"> • P_Intlk • P_Perm 	(RA-LIB)PS_nPos_4_10-00_ROUTINE.L5X
	Discrete n-Position (up to 8) Device (P_nPos)	<ul style="list-style-type: none"> • P_IntlkAdv • P_Perm 	(RA-LIB)PS_nPos_IntlkAdv_4_10-00_ROUTINE.L5X

Notes:

Numerics

2-state valve statistics (P_ValveStats)
281
alarms 320
execution 320

A

additional resources 14
advanced analog input (P_AInAdv)
16
alarms 32
execution 34
functional diagram 30

alarms

2-state valve statistics 320
advanced analog input 32
analog fanout 86
analog input 23
analog input channel 27
analog output 66
analog/pulsed control valve 286
boolean logic with snapshot 394
central reset 368
common alarm block 374
common mode block 386
condition gate delay 337
deadband controller 94
discrete 2-, 3-, or 4-state device
362
discrete input 49, 56
discrete output 59
dual sensor analog input 37
e1 plus electronic overload relay
253
e3/e3plus overload relay 259
flowmeter dosing 102
hand-operated motor 141
hand-operated valve 292
high or low selector 90
interlocks with first out and bypass
343, 348
lead/lag/standby motor group 114
mix-proof valve 301, 309
motor-operated valve 297
multiple analog input 41
n-position device 326
permissives with bypass 354
PIDE 80
powerflex 523/525 drives 146
powerflex 6000 drive 196
powerflex 7000 drive 212
powerflex 753 drive 161
powerflex 755 drive 179
pressure/temperature flow 69
restart inhibit for large motor 274
reversing motor 137
runtime and start counter 271
sequencer 98
single-speed motor 129
smc flex smart motor controller
233
smc-50 smart motor controller 227
solenoid-operated alarms 316
tank strapping table 73
two-speed motor 133
variable-speed drive 241
analog fanout (P_Fanout) 77
alarms 86
execution 86
analog input (P_AIn) 15
alarms 23
execution 23
functional diagram 21
analog input channel (P_AIChan) 16
alarms 27
execution 28

analog output (P_AOut) 19

- alarms 66
- execution 66
- functional diagram 63

analog/pulsed control valve (P_ValveC) 279

- alarms 286
- execution 287
- functional diagram 285

appendix A

- modes 398

associated tags

- common mode block 382

Autotune

- setup 80

B

boolean logic with snapshot**(P_Logic) 335**

- alarms 394
- execution 394
- functional diagram 391

C

central reset (P_Reset) 333

- alarms 368
- execution 368

common alarm block (P_Alarm) 334

- alarms 374
- execution 375

common mode block (P_Mode) 334

- alarms 386
- associate tags 382
- execution 388
- functional diagram 378
- modes 380
- operator mode 382

condition gate delay (P_Gate) 331

- alarms 337
- execution 337

configuring logic in P_Logic instance 393**cross functional**

- purpose 331
- when not to use 331
- when to use 331

D

deadband controller (P_DBC) 78

- alarms 94
- execution 94
- functional diagram 93

diagram

- lead/lag/standby 111

discrete 2-, 3-, or 4-state device**(P_D4SD) 333**

- alarms 362
- execution 362

discrete input (P_DIn)

- enablin false implementation 52

discrete input object (P_DIn) 17

- alarms 49, 56
- execution 49, 56
- functional diagram 46, 58, 63, 68, 72, 76, 79, 84, 89, 92, 96, 99, 107, 127, 132, 136, 140, 143, 158, 174, 192, 204, 224, 230, 236, 252, 257, 262, 269, 273, 283, 291, 295, 300, 314, 319, 323, 336, 340, 353, 359, 367, 371, 378, 389, 391

discrete output (P_DOut) 18

- alarms 59
- execution 60

dual sensor analog input**(P_AInDual) 17**

- alarms 37
- execution 38
- functional diagram 35

E

e1 plus electronic overload relay**(P_E1PlusE) 124**

- alarms 253
- execution 254

e3/e3 plus overload relay (P_E3Ovld)

- 125

- alarms 259
- execution 259
- required overload setup 258

e300 overload relay (P_E300Ovld)

- 125

- required overload setup 263

execution

- 2-state valve statistics 320
- advanced analog input 34
- analog fanout 86
- analog input 23
- analog input channel 28
- analog output 66
- analog/pulsed control valve 287
- boolean logic with snapshot 394
- central reset 368
- common alarm block 375
- common mode block 388
- condition gate delay 337
- deadband controller 94
- discrete 2-, 3-, or 4-state device 362
- discrete input 49, 56
- discrete output 60
- dual sensor analog input 38
- e1 plus electronic overload relay 254
- flowmeter dosing 104
- hand-operated motor 142
- hand-operated valve 293
- high or low selector 90
- interlocks with first out and bypass 343, 349
- lead/lag/standby motor group 115
- mix-proof valve 302, 310
- motor-operated valve 298
- multiple analog input 43
- n-position device 328
- permissives with bypass 355
- PIDE 82
- powerflex 523/525 drive 147
- powerflex 6000 drive 197
- powerflex 7000 drive 213
- powerflex 753 drive 163
- powerflex 755 drive 180
- pressure/temperature flow 69
- restart inhibit for large motor 275
- reversing motor 138
- sequencer 98
- single-speed motor 130
- smc flex smart motor controller 234
- smc-50 smart motor controller 228
- solenoid-operated valve 318
- tank strapping table 74
- two-speed motor 134
- variable-speed drive 243

F**fanout**

- alarms 86
- execution 86

flowmeter dosing (P_DoseFM) 78

- alarms 102
- execution 104
- functional diagram 99

functional diagram

- advanced analog input 30
- analog input 21
- analog output 63
- analog/pulsed control valve 285
- boolean logic with snapshot 391
- common mode block 378
- discrete input 46, 58, 63, 68, 72, 76, 79, 84, 89, 92, 96, 99, 107, 127, 132, 136, 140, 143, 158, 174, 192, 204, 224, 230, 236, 252, 257, 262, 269, 273, 283, 291, 295, 300, 314, 319, 323, 336, 340, 353, 359, 367, 371, 378, 389, 391
- dual sensor analog input 35
- flowmeter dosing 99
- interlocks with first out and bypass 340
- n-position device 323
- restart inhibit 273
- runtime and start counter 269

H**hand-operated motor (P_MotorHO)**

- 120

- alarms 141
- execution 142

hand-operated valve (P_ValveHO)

- 280

- alarms 292
- execution 293

HART analog input (P_AInHART)

- 20

HART analog output

- (P_AOutHART) 20

high or low selector (P_HiLoSel) 78

- alarms 90
- execution 90

I**I/O processing**

- purpose 15
- when not to use 15
- when to use 15

instruction

- long integer 401
- time and date 405

interlocks with first out and bypass

- (P_Intlk) 331

- alarms 343, 348
- execution 343, 349
- functional diagram 340
- primary operations 340, 341

L

lead/lag/standby motor group (P_LLS) 79, 107

alarms 114
execution 115
functional diagram 111

long integer

instructions 401

M

mix-proof valve (P_ValveMP) 280

alarms 301, 309
execution 302, 310

modes

common mode block 380
library objects 398

motor-operated valve (P_ValveMO)

280

alarms 297
execution 298

multiple analog input (P_AInMulti)

17

alarms 41
execution 43

N

n-position device (P_nPos) 282

alarms 326
execution 328
functional diagram 323

O

operation

deadband controller 93
interlocks 341
prompt 390

operator

common mode block 382

operator prompt (P_Prompt) 334

operation 390

P

permissives with bypass (P_Perm)

332

alarms 354
execution 355
functional diagram 353

PIDE 77

alarms 80
execution 82

powerflex 523/525 drives (P_PF52x)

121

alarms 146
execution 147
required drive set up 144

powerflex 6000 drive (P_PF6000) 122

alarms 196
execution 197

powerflex 7000 drive (P_PF7000) 122

alarms 212
execution 213

powerflex 753 drive (P_PF753) 121

alarms 161
execution 163

powerflex 755 drive (P_PF755) 122

alarms 179
execution 180

pressure/temp compensated flow (P_PTCmp) 19

alarms 69
execution 69

Process Strategies

available strategies 417
description 417

R

reference

manual scope 11

regulatory/procedural control

purpose 77
when not to use 77
when to use 77

restart inhibit for large motor

(P_ResInh) 126

alarms 274
execution 275
functional diagram 273

reversing motor (P_MotorRev) 120

alarms 137
execution 138

rules for set-reset gate 393

runtime and start counter

(P_RunTime) 125

alarms 271
functional diagram 269

S

scope

reference manual 11

sequencer object (P_Seq) 78

alarms 98
execution 98

single-speed motor (P_Motor) 119

alarms 129
execution 130

smc flex smart motor controller

(P_SMCFlex) 123

alarms 233
execution 234

smc-50 smart motor controller

(P_SMC50) 123

alarms 227
execution 228

solenoid-operated valve
(P_ValveSO) 281
alarms 316
execution 318

T

tags
common mode block 382
tank strapping table (P_StarTbl) 19
alarms 73
execution 74
time and date
instructions 405
two-speed motor (P_Motor2Spd) 120
alarms 133
execution 134

V

valves
purpose 279
when not to use 279
when to use 279
variable-speed drive (P_VSD) 124
alarms 241
execution 243
required connections 238

Notes:

Rockwell Automation Support

Use the following resources to access support information.

Technical Support Center	Knowledgebase Articles, How-to Videos, FAQs, Chat, User Forums, and Product Notification Updates.	https://rockwellautomation.custhelp.com/
Local Technical Support Phone Numbers	Locate the phone number for your country.	http://www.rockwellautomation.com/global/support/get-support-now.page
Direct Dial Codes	Find the Direct Dial Code for your product. Use the code to route your call directly to a technical support engineer.	http://www.rockwellautomation.com/global/support/direct-dial.page
Literature Library	Installation Instructions, Manuals, Brochures, and Technical Data.	http://www.rockwellautomation.com/global/literature-library/overview.page
Product Compatibility and Download Center (PCDC)	Get help determining how products interact, check features and capabilities, and find associated firmware.	http://www.rockwellautomation.com/global/support/pcdc.page

Documentation Feedback

Rockwell Automation maintains current product environmental information on its website at <http://www.rockwellautomation.com/rockwellautomation/about-us/sustainability->

1336 PLUS, Allen-Bradley, E1 Plus, E300, FactoryTalk, Logix 5000, PlantPAx, PowerFlex, Rockwell Software, Rockwell Automation, SMC, Studio 5000 Logix Designer, and TotalFORCE are trademarks of Rockwell Automation, Inc.

Rockwell Otomasyon Ticaret A.Ş., Kar Plaza İş Merkezi E Blok Kat:6 34752 İçerenköy, İstanbul, Tel: +90 (216) 5698400

www.rockwellautomation.com

Power, Control and Information Solutions Headquarters

Americas: Rockwell Automation, 1201 South Second Street, Milwaukee, WI 53204-2496 USA, Tel: (1) 414.382.2000, Fax: (1) 414.382.4444

Europe/Middle East/Africa: Rockwell Automation NV, Pegasus Park, De Kleetlaan 12a, 1831 Diegem, Belgium, Tel: (32) 2 663 0600, Fax: (32) 2 663 0640

Asia Pacific: Rockwell Automation, Level 14, Core F, Cyberport 3, 100 Cyberport Road, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846

Publication PROCES-RM013I-EN-P - April 2023

Supersedes Publication PROCES-RM013H-EN-P - December 2021

Copyright © 2023 Rockwell Automation, Inc. All rights reserved. Printed in the U.S.A.