



PlantPAx Display and Library Guidelines

Process Library 5.20

Process Library 5.30



Allen-Bradley

by ROCKWELL AUTOMATION

Reference Manual

Original Instructions

Important User Information

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

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

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

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

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

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

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

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



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



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

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

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



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



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



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

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



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

	Preface	11
	About This Publication	11
	Summary of Changes	11
	Additional Resources	11
	Chapter 1	
Rockwell Automation Library of Process Objects	Rockwell Automation Services and Support	14
	Library Versions	15
	Tag Extended Properties and Default Alarm Settings	16
	Command Sources and Device Virtualization	17
	Command Sources	17
	Virtualization	18
	Faceplate Alarms	19
	Faceplate Alarm Banner (Collapsed Banner)	19
	Faceplate Alarm Banner (Expanded Banner)	20
	Linear Gauge Alarm Colors	20
	Alarm Tab Border Color	21
	Alarm Tab – Full Alarm Management	21
	Alarm Severity Colors and Behavior	22
	PlantPAx Process Library Migration Tool	22
	Chapter 2	
Organization, Ownership, Arbitration, and Propagation (OOAP)	Overview	23
	Propagation	25
	Propagated Commands	25
	Propagated Status	27
	Bus Faceplates	28
	Operator Tab	28
	Maintenance Tab	29
	Alarms Tab	29
	Bus Ownership	29
	Ownership Configuration	33
	Request Ownership	36
	Unbinding Ownership	38
	Exclusion	39
	Workflow	40
	Manually Configure Organizations	41
	Create the Organization Logic	41
	Example Logic	45
	Define the Bus Elements	45
	Configure the Area Instance	47
	Configure the Unit Instances	47
	Configure the Equipment Phase or Equipment Module Instances	48
	Configure the Device Instances to Use the Bus Elements	48
	Add Devices	50
	Define the OrgView Elements	50

Example OrgView Elements	51
Create the Organizational Tree in the HMI Client	51
Configure the Client Display	52
Build the Node Tree (FactoryTalk View)	53
Build the Node Tree (FactoryTalk Optix)	59
Set Start Node	64
Node Array Guidelines	64
Node Tooltip Information	65
Status Indicators	65
Arbitration	66
Unit Group Control	69
Unit Group Object Logic Example	71
Hardware Organization Bus	72
Create the Hardware Organization Logic	74
Create the HardwareTree Program	74
Create Array Tags	74
Define the HWBus Elements	76
Configure the LCPU Instance	76
Configure the Module Status Instances	77
Configure the Task Monitor Instances	78
Create the Organizational Tree in the HMI Client	79
Configure the Client Display	79
Hardware Tree Alarm Gating	80

Chapter 3

Ownership (raP_Opr_Owner)

Guidelines	81
Functional Description	81
Required Files	82
Controller Files	82
Visualization Files	82
Operations	82
Command Sources	82
Alarms	82
Virtualization	82
Execution	82
Programming Examples	82

Chapter 4

Arbitration (raP_Opr_ArbitrationQ)

Guidelines	85
Functional Description	85
Required Files	86
Controller Files	86
Visualization Files	86
Operations	86
Command Sources	86
Alarms	86
Virtualization	86
Execution	86
Tag Extended Properties and Default Alarm Settings	86

	Programming Examples.....	87
Organizational Scan (raP_Opr_OrgScan)	Chapter 5	
	Guidelines	89
	Functional Description.....	89
	Required Files	89
	Controller Files	90
	Visualization Files.....	90
	Operations	90
	Command Sources	90
	Alarms.....	90
	Virtualization.....	90
	Execution.....	90
	Tag Extended Properties and Default Alarm Settings	90
	Programming Examples.....	90
Organizational View (raP_Opr_OrgView)	Chapter 6	
	Guidelines	91
	Functional Description.....	91
	Required Files	92
	Controller Files	92
	Visualization Files.....	92
	Operations	92
	Command Sources	92
	Alarms.....	92
	Virtualization.....	92
	Execution.....	92
	Tag Extended Properties and Default Alarm Settings	92
Organizational Node Interface (raP_Opr_OrgDeviceCtrl)	Chapter 7	
	Guidelines	93
	Functional Description.....	94
	Required Files	95
	Controller Files	95
	Visualization Files.....	95
	Operations	95
	Command Sources	95
	Alarms.....	95
	Virtualization.....	95
	Execution.....	95
	Programming Examples.....	96
	Shared Child Device Example	96
	Selectively Owning a Parent	97
	Limitations.....	100

Process Area Module (raP_Opr_Area)

Chapter 8

Guidelines	101
Functional Description	102
Command Source Management	102
Required Files	102
Controller Files	102
Visualization Files	102
Operations	102
Command Sources	102
Alarms	102
Virtualization	103
Execution	103
Programming Example	103

Process Unit (raP_Opr_Unit)

Chapter 9

Guidelines	105
Functional Description	106
Command Source Management	106
Required Files	106
Controller Files	106
Visualization Files	106
Operations	106
Command Sources	106
Program Structure	107
Alarms	107
Virtualization	107
Execution	108
Local Message	108
FactoryTalk Optix Local Message	108
Tag Extended Properties and Default Alarm Settings	109
Programming Example	110

Generic Equipment Module (raP_Opr_EMGen)

Chapter 10

Guidelines	111
Functional Description	112
Required Files	113
Controller File	113
Visualization Files	113
Operations	113
Command Sources	113
State Model	114
Program Structure	114
Alarms	114
Virtualization	115
Execution	115
Local Message	115
FactoryTalk Optix Local Message	116
Tag Extended Properties and Default Alarm Settings	117

	Programming Example.....	118
Variable State Machine (raP_Opr_VSM)	Chapter 11	
	Functional Description.....	121
	Permissive.....	121
	Interlock.....	122
	Command Source.....	122
	State Complete.....	123
	Idle State.....	123
	Auxiliary Commands.....	123
	State Machine Configuration (raP_UDT_Opr_VSMCfgr).....	123
Generic Equipment Phase (raP_Opr_EPGen)	Chapter 12	
	Guidelines.....	127
	Functional Description.....	128
	Required Files.....	129
	Controller File.....	129
	Visualization Files.....	129
	Operations.....	129
	Command Sources.....	129
	Phase Manager.....	129
	Program Structure.....	130
	Alarms.....	131
	Virtualization.....	131
	Execution.....	131
	Local Message.....	131
	FactoryTalk Optix Local Message.....	132
	Tag Extended Properties and Default Alarm Settings.....	133
	Programming Example.....	133
Parameter and Reports (raP_Tec_ParRpt)	Chapter 13	
	Guidelines.....	135
	Functional Description.....	136
	Required Files.....	137
	Controller File.....	137
	Visualization Files.....	137
	Operations.....	137
	Command Sources.....	137
	Alarms.....	137
	Virtualization.....	137
	Execution.....	138
	Tag Extended Properties and Default Alarm Settings.....	138
	Programming Example.....	139
	Parameter Program Example.....	139
	Reports Program Example.....	140

Operator Prompt (raP_Opr_Prompt)

Chapter 14

Guidelines	141
Functional Description	141
Required Files	142
Controller Files	142
Visualization Files	142
Operations	142
Command Sources	142
Alarms	142
Virtualization	142
Tag Extended Properties and Default Alarm Settings	143

Logix Diagnostic Objects

Chapter 15

Logix Change Detector (raP_Dvc_LgxChangeDet)	145
Guidelines	145
Functional Description	145
Required Files	146
Operations	147
Programming Example	148
Logix Controller CPU Utilization (raP_Dvc_LgxCPU_5x80)	150
Guidelines	151
Functional Description	151
Required Files	152
Operations	152
Programming Example	153
Logix Redundant Controller Monitor (raP_Dvc_LgxRedun)	156
Guidelines	156
Functional Description	156
Required Files	156
Operations	157
Programming Example	158
Logix Module Status (raP_Dvc_LgxModuleSts)	161
Guidelines	161
Functional Description	161
Required Files	162
Operations	162
Programming Examples	163
Logix Task Monitor (raP_Dvc_LgxTaskMon)	165
Guidelines	165
Functional Description	165
Required Files	166
Operations	166
Programming Example	167
Logix Event (raP_Tec_LgxEvent)	167
Guidelines	167
Functional Description	167
Required Files	169
Operations	169
Programming Examples	170

Process Extended Alarms (raP_Opr_ExtddAlm)	Chapter 16 Functional Description..... 173 Required Files 174 Controller Files 174 Visualization Files..... 174 Operations..... 175 Command Sources 175 Alarms..... 175 Virtualization 175 Execution..... 175 Programming Examples..... 176 Implementation by Using the EnableIn False Feature 176
5094-IF8IH to PAH Configuration Example	Appendix A Download and install the 5094 HART Analog Add-On Profile 177 Add the 5094 Adapter Module to the Project I/O Configuration 179 Add the 5094-IF8IH Module to the Project I/O Configuration 180 Add the HART Device to the Project I/O Configuration..... 181 Configure the Analog Input Channel..... 184 Add the PAH (Process Analog HART) and PAI (Process Analog Input) Instruction Instances to the Project. 185 Add the PAH Instruction Instance 185 Connect PAX_HART_DEVICE:I:O Member from Input Assembly to Ref_HARTData InOut Parameter..... 187 Add the PAI Instruction Instance..... 187 Connect the PAH Instance to the PAI Instance 190
1756-IF8IH with raP_Tec_HARTChanData_to_PA H Add-On Instruction Configuration Example	Appendix B Add the 1756-IF8IH Module to the Project I/O Configuration 195 Configure the Channel for the HART Device 197 Import the raP_Tec_HARTChanData_to_PAH Add-On Instruction 198 Import the I_1756IF8IH Rung into the Project 199 Add the raP_Tec_HARTChanData_to_PAH Instance to the Project..... 203 Add the PAH and PAI Instances to the Project and Connect PAH and PAI Instances . 207

Notes:

About This Publication

This publication Describes the PlantPAx® Add-On Instructions that are available to develop applications.

Summary of Changes

This publication contains the following new or updated information. This list includes substantive updates only and is not intended to reflect all changes.

Topic	Page
Moved graphic framework chapters to PROCES-RM251	
Moved FactoryTalk View faceplates to PROCES-RM250	
Moved FactoryTalk Optix faceplates to PROCES-RM260	
Added Extended alarms chapter	171
Removed Studio 5000 View Designer® support	Throughout

Additional Resources

These documents contain additional information concerning related products from Rockwell Automation. You can view or download publications at rok.auto/literature.

Resource	Description
PlantPAx Process Solutions Technical Documentation	Quickly access and download technical specifications, installation instructions, and user manuals.
Configuration and Implementation User Manual, publication PROCES-UM100	Provides system guidelines and instructions to assist with the development of your PlantPAx system.
Rockwell Automation Sequencer Object, Publication PROCES-RM202	Provides an overview of how to use the Rockwell Automation Sequencer Object. The manual includes a Sequencer programming demonstration, example, and configuration instructions.
FactoryTalk View Display Implementation Guidelines, publication PROCES-RM250	Describes the PlantPAx Process instructions, and associated FactoryTalk View faceplates that are available to develop applications.
FactoryTalk Optix Display Implementation Guidelines, publication PROCES-RM260	Describes the PlantPAx Process instructions, and associated FactoryTalk Optix faceplates that are available to develop applications.
PlantPAx Process Control Instructions, publication PROCES-RM215	This manual provides a programmer with details about the available Process instruction set for a Logix-based Process controller.
Graphic Framework Guidelines, publication PROCES-RM251	This publication Describes the PlantPAx graphic framework.
Process Object parameters Spreadsheet, publication, PROCES-RD200	Describes the PlantPAx Process object parameters.
PlantPAx Visualization Files, publication, PROCES-RD201	Describes the visualization files that are required for the Library of Process Objects.
FactoryTalk Optix Solutions, OPTIX-AT001	Provides an overview of the system, application examples, and ordering guidelines to help you choose exactly what you need. It also guides you through the basics of creating and deploying your own application.
System Security Design Guidelines Reference Manual, publication SECURE-RM001	Provides guidance on how to conduct security assessments, implement Rockwell Automation products in a secure system, harden the control system, manage user access, and dispose of equipment.
Safety Guidelines for the Application, Installation, and Maintenance of Solid-state Control, publication SGI-1.1	Designed to harmonize with NEMA Standards Publication No. ICS 1.1-1987 and provides general guidelines for the application, installation, and maintenance of solid-state control in the form of individual devices or packaged assemblies incorporating solid-state components.
Industrial Automation Wiring and Grounding Guidelines, publication 1770-4.1	Provides general guidelines for installing a Rockwell Automation industrial system.
ProposalWorks™ configuration software, rok.auto/systemtools	Helps configure complete, valid catalog numbers and build complete quotes based on detailed product information.
Rockwell Automation Global SCCR tool, rok.auto/sccr	Provides coordinated high-fault branch circuit solutions for motor starters, soft starters, and component drives.
Product Certifications website, rok.auto/certifications	Provides declarations of conformity, certificates, and other certification details.

Notes:

Rockwell Automation Library of Process Objects

The Rockwell Automation® Library of Process Objects, also referred to in this document as the Process Library, contains the tools that are required to enable consistent deployment and faster product delivery.

The Library Includes the following:

- Graphics for built-in instructions
- HMI images and Help files
- Logix diagnostic objects
- Process objects
- Premier Integration objects
- Control strategies
- Sequencer objects
- Color Change tool
- Historian -- Asset Framework template and objects

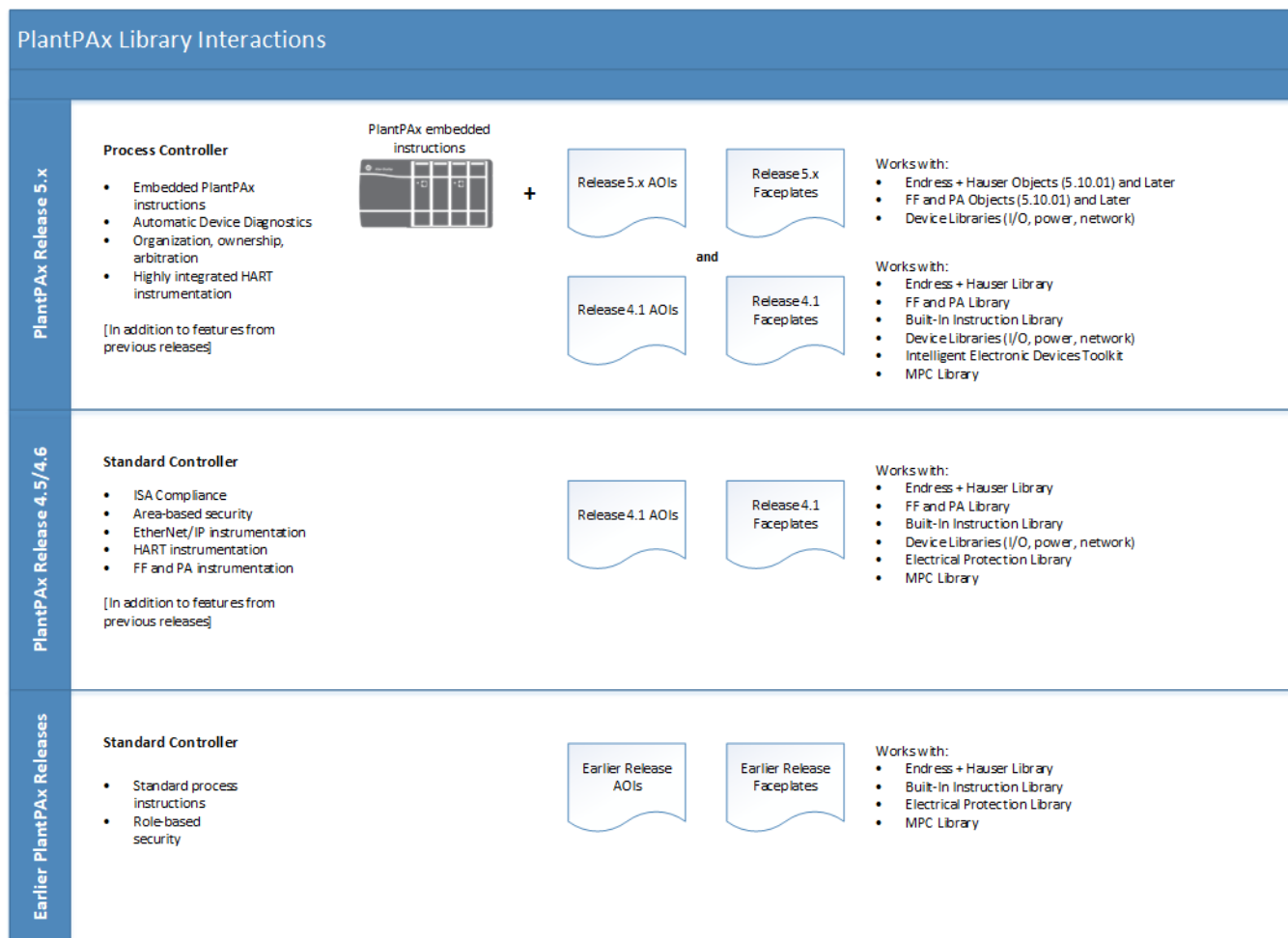
The Library of Process objects is designed to work in conjunction with the following libraries:

Item	Description
I/O Device Library	Provides objects for Rockwell Automation 1756, 1769, 1734, 1794, 1738, 1732E, 1719, 5069, 5094 I/O modules. Provides preconfigured status and diagnostic faceplates sets for Rockwell Automation digital and analog I/O devices. You can use these objects with Machine Builder, Process, and Packaged Libraries, or as standalone components.
IO-Link Device Library	Provides IO-Link master and sensor objects. Provides preconfigured status and diagnostic faceplates.
Machine Builder Libraries	Library objects for use with Application Code Manager. <ul style="list-style-type: none"> • Independent Cart Technology Libraries, includes ICT Libraries for iTRAK® and MagneMotion® • Studio 5000® Application Code Manager • Power Device Library, including objects for E300, ArmorStart®, PowerFlex®, and Kinetix®
Network Device Library	Provides objects for Stratix® switch and Device Level Ring network objects.
Power Device Library	Provides objects for discrete and velocity power devices.

In addition to the libraries, the Intelligent Electronic Devices Toolkit contains Add-On Instructions and visualization objects. The Intelligent Electronic Devices Toolkit supports communication via the ProSoft Technology IEC 61850 communication module, MVI56E-61850C, and the ProSoft Technology EtherNet/IP™ Server to IEC 61850 Dual Port Client Gateway, PLX82-EIP-61850. See Rockwell Automation Intelligent Electronic Devices Toolkit, publication [PROCES-RM211](#) for more information.

When you deploy the process controller in PlantPAx® 5.0 and later, you gain access to additional PlantPAx instructions. The PlantPAx instructions on the process controller provide objects that are embedded in the controller firmware. For more information on faceplates for these instructions, see FactoryTalk View Display Implementation Guidelines, publication [PROCES-RM250](#) and FactoryTalk Optix Display Implementation Guidelines, publication [PROCES-RM260](#).

See Logix 5000® Advanced Process Control and Drives Instructions, publication [1756-RM006](#) for more information on PlantPAx Instructions.



Rockwell Automation Services and Support

System Support offers technical assistance that is tailored for control systems. Some of the features include the following:

- Highly experienced team of engineers with training and systems experience
- Process support at a systems-level that is provided by process engineers
- Use of online remote diagnostic tools
- Access to otherwise restricted TechConnectSM Knowledgebase content
- 24-hour, 7 days per week, 365 days per year of phone-support coverage upgrade option

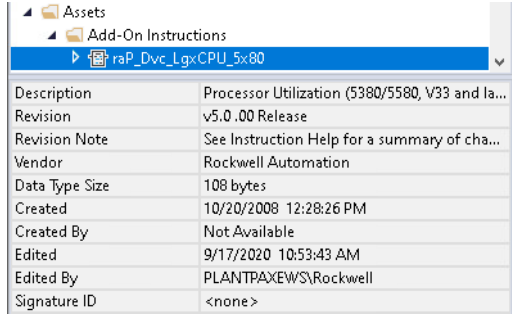
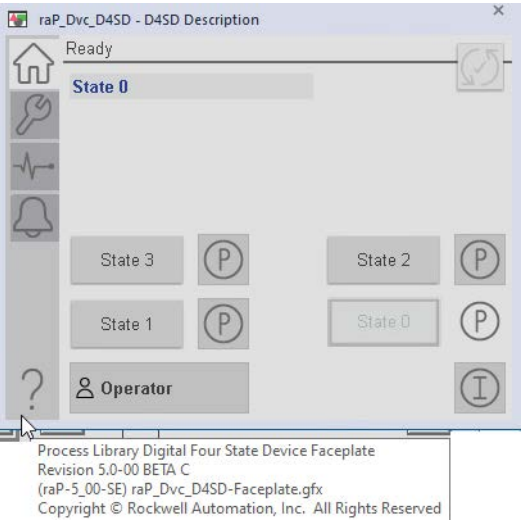
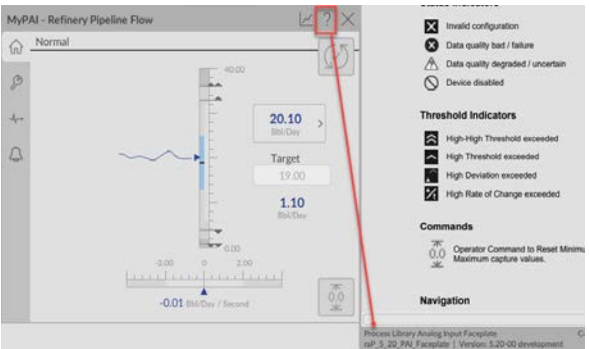
For more information, contact your local distributor or Rockwell Automation representative or see <https://www.rockwellautomation.com>.

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

Libraries can be accessed from the [Product Compatibility and Download Center](#).

Library Versions

Each library object has a revision x.yy.zz where: x is the Major Revision number, yy is the Minor Revision number, and zz is the Maintenance Release. Each release of the Process Library comes with release notes that describe the changes that were made since the last release.

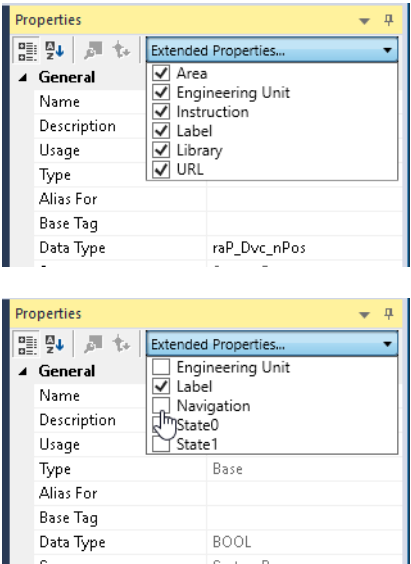
Component	Example
The Add-On Instruction in Logix Designer application has revision information visible when the instruction is selected in the Controller Organizer.	
The faceplate in FactoryTalk View software has revision information visible when the pointer is paused just inside the lower left corner of the faceplate.	
The faceplate for FactoryTalk Optix software has revision information visible at the bottom of the Help window.	

Tag Extended Properties and Default Alarm Settings

Tag extended properties must be configured to drive the text on the operations faceplate. See Logix 5000 Controllers I/O and Tag Data, publication [1756-PM004](#) for more information on extended tags.

Access to alarms is via <backing_tag>.@Alarms.<alarm_name>.<alarm_parameter>.

You must select the extended properties to populate for each tag and then enter the values.



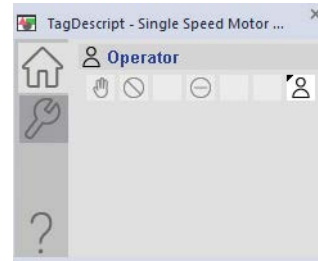
Command Sources and Device Virtualization

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.

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 screenshot on the icon in the far right) indicates the “Normal” command source.



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_OvrCmd) and other Override settings are accepted. If so configured (for example, Cfg_OvrPermIntlk=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.

Not all Command Sources are used in every object.

PCMSRC	Operator	Program	External	Override	Maintenance	Out of Service	Hand
raP_Opr_Area	x	x	x		x	x	
raP_Opr_Unit	x	x	x		x	x	
raP_Opr_EMGen	x	x	x		x	x	
raP_Opr_EPGen	x	x	x		x	x	

Virtualization

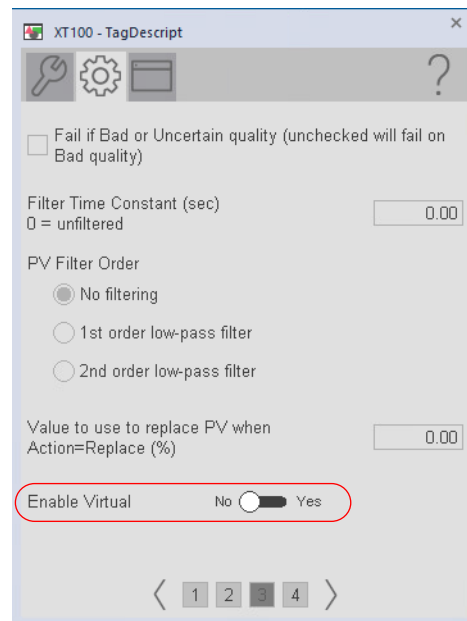
Virtualization is used with device objects to simulate operation of a device instead of controlling the actual device. Virtualization is used for such activities as system testing or operator training, where the process is shut down or not connected to the controller.

When a device is set to Physical operation, the actual field device I/O is monitored or controlled, and the field device operates normally, on-process.

When a device object is set to Virtual operation, the I/O for the field device are ignored, and the device operates in one of these manners:

- For monitored devices, such as analog and discrete inputs, a virtual process variable (PV) is provided, either by simulation logic or by entry from the HMI faceplate.
- For controlled devices, such as valves, motors, and drives, the outputs are held de-energized (at zero) and the object responds in a "loopback" manner, as if an actual device were connected. So a valve object, while keeping outputs de-energized, reports valve status to the operator and to program logic as if the valve were opening and closing normally.

To select Virtual or Physical operation, go to the Advanced faceplate for the device and toggle the Virtual / Physical selector.



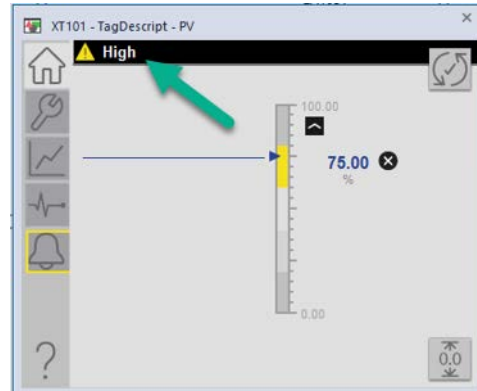
Faceplate Alarms

PlantPAx Process Library faceplates use multiple visual mechanisms to ensure that operators can quickly recognize abnormal conditions, identify alarm severity, and understand which alarms require action. Each mechanism serves a specific operational purpose and works together with others to create a consistent and intuitive alarm experience.

Faceplate Alarm Banner (Collapsed Banner)

The alarm banner appears at the top of the faceplate and provides an immediate summary of the most important alarm condition.

This banner is purposely attention grabbing, ensuring operators never overlook conditions that require immediate confirmation, for more details you can always navigate to the alarm tab.



If any alarms require acknowledgment, the banner highlights one of those alarms.

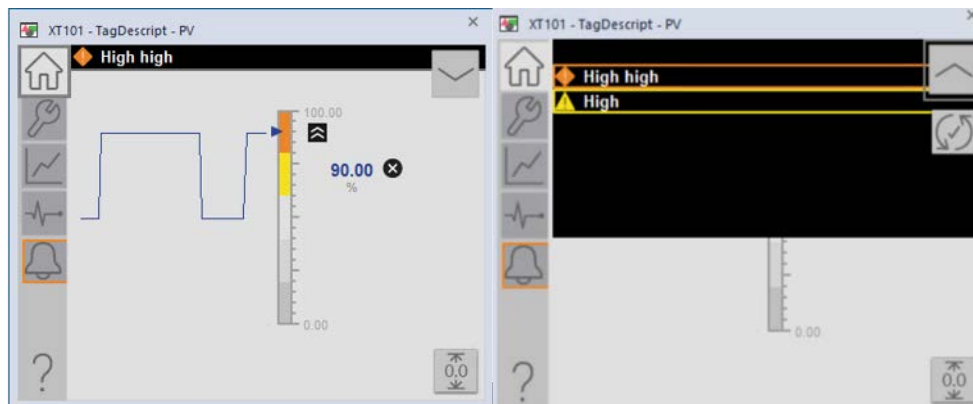
If no acknowledgment is needed but active alarms exist, the banner draws attention to one of the active alarms.

If multiple alarms exist:

- The Alarms that require acknowledgment or are active are visible, they are layered in a predetermined order, and only one is displayed. The predetermined order for PAI (first in the list shows on top of items lower in the list):
 - Failure Alarm
 - High High Alarm
 - High Alarm
 - Low Alarm
 - Low Low Alarm
 - Hi Deviation Alarm
 - Low Deviation Alarm
 - High Rate of Change Alarm
- The banner indicates an icon to the operator for them to expand the banner for more details.

Faceplate Alarm Banner (Expanded Banner)

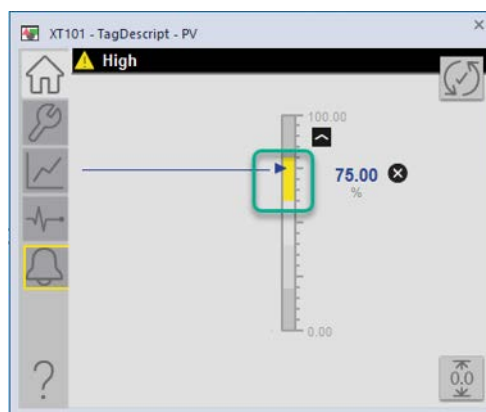
When the operator expands the banner, alarms that require acknowledgment are shown. This expanded view is intended for an operator for a quick, focused review of unacknowledged alarms.



Linear Gauge Alarm Colors

The process value (PV) gauge is visually enhanced to show alarm conditions directly within the measurement display. Active alarms are color-coded directly on the gauge which helps operators visually match the process condition with alarm thresholds.

The color-coded gauges provide an intuitive, at a glance understanding of how far the PV is beyond its normal range.



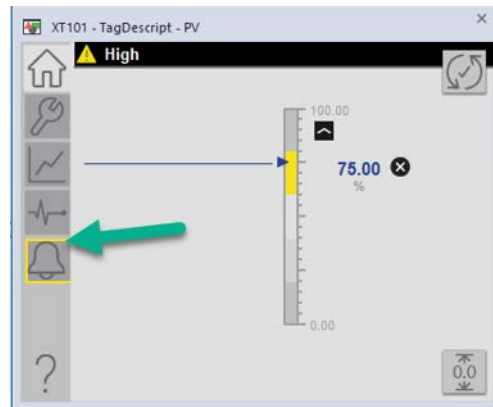
Alarm Tab Border Color

The alarm tab itself changes appearance to help operators see, from any faceplate tab, that alarm conditions exist.

The tab border color changes to reflect the active alarm severity. This draws attention even when the alarm banner is collapsed or the operator is in another tab.

This provides a persistent visual cue across all faceplate pages.

This graphical method reinforces alarm visibility regardless of whether the operator has opened the alarm tab.

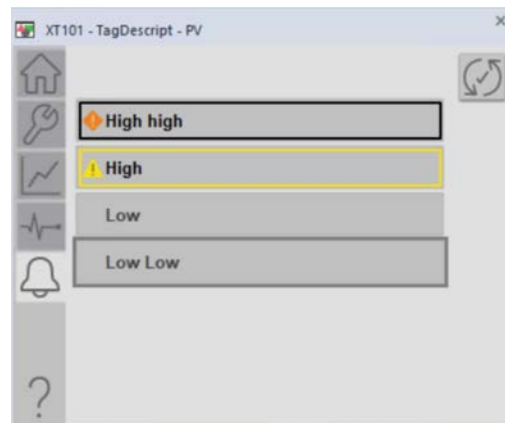


Alarm Tab - Full Alarm Management

The Alarm tab provides the complete list of:

- All configured alarms
- Their status
- The ability to configure alarms (depending on user permissions)

Operators use this tab to manage alarm details beyond the summary provided in the banner.



Alarm Severity Colors and Behavior

The Process Library uses a defined color scale to communicate the severity of alarms:

Severity Range	Severity Level	Color
1...250	Low	Purple
251...500	Medium	Yellow
501...750	High	Orange
751...1000	Urgent	Red

- The border or shape shade displays the color of the highest severity active alarm.
- Flashing occurs when any alarm requires acknowledgment, making it exceptionally clear to the operator.

This ensures that the most critical condition always takes precedence in the operator's view.

PlantPax Process Library Migration Tool

This tool is used to migrate from previous Process Library versions to version 5.30. The PlantPax Process Library Migration Tool provides the following:

- Updates Logix controller ACD files containing Rockwell Automation Process Library Add-On Instruction tags to corresponding Process Controller predefined process instruction tags and version 5.30 Add-On Instruction tags.
- Converts FactoryTalk View SE process graphics XML files containing global object references from previous Process Library versions to version 5.30 Process Library global objects.
- Migration of Process Library HMI libraries.
- Migration of GEMS Version 4.4 Add-On Instruction to corresponding Process Controller predefined process instruction tags and version 5.30 Add-On Instruction tags.

The tool reduces engineering time and migration errors. Use the tool to keep up with the latest Rockwell Automation software features and increase the lifecycle of the PlantPax DCS.

Organization, Ownership, Arbitration, and Propagation (OOAP)

Overview

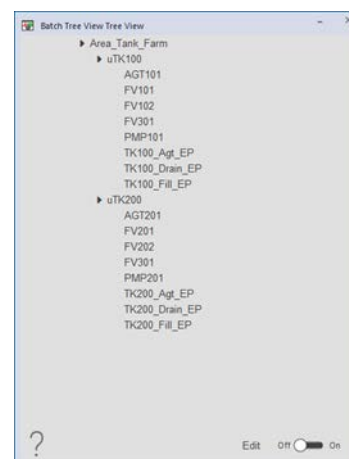
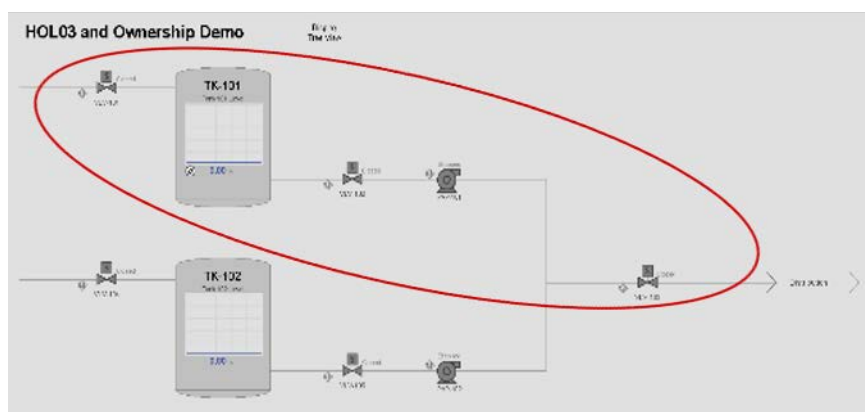
When should you use the OOAP functionality?

- A hierarchy exists among the equipment in the system
- Shared devices among equipment groups
- Design includes the use of Add-On Instruction process objects Area, Unit, Equipment Phase, Equipment Module, or Sequencer. These objects provide additional functionality when configured as parents in the organizational trees.
 - Show tree view faceplate navigation function from object faceplates provides a quick visualization of the state of an equipment group for troubleshooting.
 - PCmd_OwnerAcq and PCmd_OwnerRel interface from these Add-On Instructions provide a standard approach to organizing the children of the equipment group for control.
- Requirements exist for monitoring the aggregated statuses of a specific equipment group for maintenance or reporting purposes
- A Secondary Hardware Organization provides consolidated I/O Module status. Control Strategies can be informed of faults along the connection path it is in.

When should you NOT use the OOAP functionality?

- All equipment operates independently

Organization is a method by which parent / child relationships can be created and modified among PlantPax® Instructions. Organization provides a method to propagate a selected subset of commands (related to command source, alarms, and so forth) from the parent down to its children or propagate the aggregate of a selected subset of status (related to command source, alarms, and so forth) from the children up to one or more parents.



Organizational views can be many nodes deep and wide, and numerous organizational views can reference the same devices to suit the needs of the user. The structure and view of these organizational trees can be modified online from the HMI by adding child nodes underneath existing nodes. The same bus object can be referenced on multiple nodes in the organization tree. When a bus object is referenced in multiple nodes in the organization, the parent node must be unique.

Organizing equipment provides the ability to coordinate commands of related equipment and view their related status (equipment modules or phase modules), or alternatively to monitor specific equipment or equipment types as a maintenance function.

A special use case of the organization function is the hardware bus that uses the `raP_Opr_LogixModuleSts` instruction to monitor the connection status of supported hardware modules. This is configured independently to provide an organizational view of a controller I/O tree configuration. By using the organizational tree for hardware, the alarms can be gated for child modules allowing root cause analysis by only alarming the highest module, which caused the alarm in the tree.

There are three basic capabilities that can be added incrementally:

Function	Description
Organization	Create parent/child relationships among objects to provide a method to propagate commands from the parent to its children and propagate status from the children to the parent.
Ownership	Uses the Organization backbone to allow a parent to take ownership of its children by placing them into Program state and accept the Owner's ID
Arbitration	Provides ability to manage and prioritize ownership of shared equipment

The implementation of these three capabilities can provide the following benefits:

- Provides the ability to command devices as an Equipment Group
- No additional Logix-based code is required to group equipment
- Consolidates information (alarms, device modes)
- Queues ownership requests and lets users set arbitration rules
- Applying standard solutions not only for direct device control, but also for device management, which enables a common look and feel throughout the complete application

The two major data structures that are the backbone of OOAP are the Bus and Node arrays. The bus is an interface to PlantPAx objects. It is used to propagate a specific set of statuses and commands through any user-defined organizational structure. A single element in the node array maintains the location of a bus object in the larger organizational tree. A single bus object can exist in multiple locations on the node array as a child. For example, a PVLV object could be a child to multiple equipment modules in the organization. The configuration for which commands and statuses are propagated between parents and children is maintained on the node array.

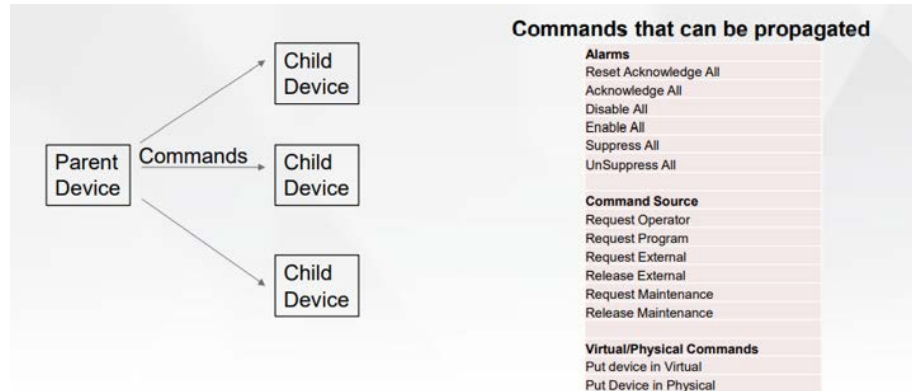
The individual elements in the bus array can be either real-time or non-real-time components.

- **Real-Time:** A bus object assigned to a controller executed PlantPAx Process Object (PVLV, PMTR, EMGen, and so on). These objects are time critical.
- **Non-Real-Time:** Elements that are purely organizational. Either for access or the aggregation of statuses. These objects are not time critical.

Propagation

The organizational trees that users create by defining parent/child relationships among equipment will immediately establish command and status propagation on the bus. Users can define which commands and statuses get propagated at each node's configuration display. Commands can be issued to children and child statuses can be viewed from the bus faceplate display. Designers can reference the bus command and status bits defined below in logic as needed. For example, Bus[Parent_Index].Obj.Inp_Cmd.14 will place all child devices in the external mode with a single command. The Inp_CmdAll.14 bit places the parent AND its children into external. It is recommended to latch and unlatch bus commands when implementing any user-defined logic with the following commands.

Propagated Commands



Commands Issued from the Parent Bus Object and Propagated to the Children in the Organization

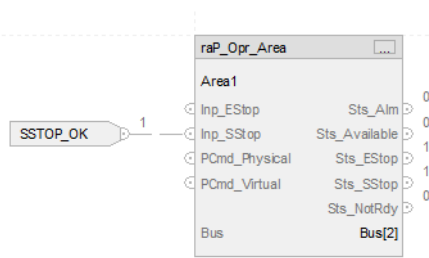
Bus[x].Obj.Inp_Cmd Bit	Command	Description
1	Reset/Ack All	Issue a Reset/Ack All to all objects
2	Reset	Issue a Reset to all objects
3	Disable alarms	Disable all alarms
4	Enable alarms	Enable all alarms
5	Suppress alarms	Suppress all alarms
6	Unsuppress alarms	Unsuppress all suppressed alarms
7	Unshelve alarms	Unshelve all shelved alarms
10	Request Virtual	Request all to be in Virtual
11	Request Physical	Request all to be in Physical
12	Request Oper	Request all to be in Operator
13	Request Prog	Request all to be in Program
14	Request Ext	Request all to be in External
15	Release Ext	Release all from External
16	Request Maint	Request all to be in Maintenance
17	Release Maint	Release all from Maintenance
28	Unit State Request 1	Unit User-Defined State Request 1
29	Unit State Request 2	Unit User-Defined State Request 2
30	Unit State Request 3	Unit User-Defined State Request 3
31	Unit State Request 4	Unit User-Defined State Request 4

Commands Issued from the Parent Bus Object that are Issued to the Parent and its Children in the Organization

Bus[x].Inp_CmdAll Bit	Command	Description
1	Reset/Ack All	Issue a Reset/Ack All to all objects
2	Reset	Issue a Reset to all objects
3	Disable alarms	Disable all alarms
4	Enable alarms	Enable all alarms
5 ⁽¹⁾	Suppress alarms	Suppress all alarms
6 ⁽¹⁾	Unsuppress alarms	Unsuppress all suppressed alarms
7	Unshelve alarms	Unshelve all shelved alarms
10	Request Virtual	Request all to be in Virtual
11	Request Physical	Request all to be in Physical
12	Request Oper	Request all to be in Operator
13	Request Prog	Request all to be in Program
14	Request Ext	Request all to be in External
15	Release Ext	Release all from External
16	Request Maint	Request all to be in Maintenance
17	Release Maint	Release all from Maintenance

(1) The Area, Unit, EMGen, and EPGen Add-On Instructions can issue a PCmd_SuppressAllAlms and PCmd_UnsuppressAllAlms command using the bus. These commands use the CmdAll bus interface bits 5 and 6 to issue the suppress or unsuppress command to the parent AND its children.

The Line Level High commands that are defined in the previous table can be utilized to stop and start equipment as a group. For example, a software stop OK input signal can be mapped to the Area.Inp_Sstop input. Any Unit object that is defined as a child underneath that Area object in the organizational tree that is also configured to accept software stop commands through the bus goes to a not ready state when the software stop input = 0.



Likewise, an Interlock not OK status from a Unit object can propagate down the organizational tree to multiple Equipment Phase objects to set a not ready status and prevent operation.

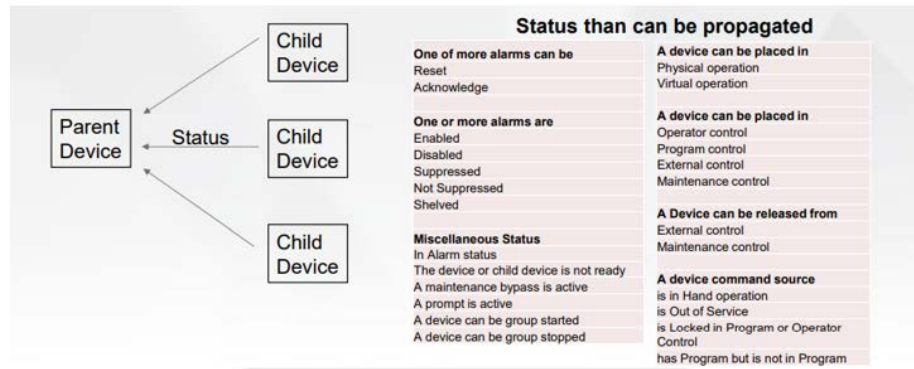
Line Level High Commands Issued and Propagated to both Parent and Child

Bus[x].Out_CmdLLH Bit	Command	Description
0	Emergency Stop ⁽¹⁾	1 = Emergency Stop OK (Inp_Estop)
1	Software Stop	1 = Software Stop OK (Inp_SStop)
2	Permissive OK ⁽²⁾	1 = Permissive OK
3	Interlock OK ⁽²⁾	1 = Interlocks OK

(1) Exists in the Area and Unit Add-On Instructions only.

(2) Exists in the Unit, EMGen, and EPGen Add-On Instructions only.

Propagated Status



Bus[x].Obj.Out_Sts Bit	Status Produced and Propagated from Child to Parent	Description
0	Alarms Active	At least one alarm is active for this object or its children
1	Alarms/Object to be Reset	At least one alarm is ready for reset for this object or its children
2	Ready for Reset	At least one object or child is ready for reset
3	Alarms Enabled	At least one alarm is enabled for this object or its children
4	Alarms Disabled	At least one alarm is disabled for this object or its children
5	Alarms Unsuppressed	At least one alarm is not suppressed for this object or its children
6	Alarms Suppressed	At least one alarm is suppressed for this object or its children
7	Alarms Shelved	At least one alarm is shelved for this object or its children
8	Object Not Ready	At least one object or child is not ready
9	Maint Bypass Active	At least one object or child has a Maint Bypass active
10	Objects in Physical	At least one object or child is in Physical
11	Objects in Virtual	At least one object or child is in Virtual
12	Ready for Oper Request	At least one object or child is ready for an Oper request
13	Ready for Prog Request	At least one object or child is ready for a Prog request
14	Ready for Ext Request	At least one object or child is ready for an Ext request
15	Ready for Ext Release	At least one object or child is ready for an Ext release request
16	Ready for Maint Request	At least one object or child is ready for a Maint request
17	Ready for Maint Release	At least one object or child is ready for a Maint release request
18	Objects in Hand	At least one object or child is in hand
19	Objects Out of Service (OoS)	At least one object or child is Out of Service
20	Objects in Oper or Prog Locked	At least one object or child is Oper or Prog Locked
21	Objects has Prog, is not in Prog	At least one object or child has Prog but is not in Prog
22	Prompt is active	At least one object or child has an active prompt
28	Unit user-defined state 1	At least one child of the Unit is not in state 1
29	Unit user-defined state 2	At least one child of the Unit is not in state 2
30	Unit user-defined state 3	At least one child of the Unit is not in state 3
31	Unit user-defined state 4	At least one child of the Unit is not in state 4



In versions 5.20 and earlier, the Bus[X].Obj.CMDLLHMask tag is used to isolate children Out_Sts aggregation from the parent status. This allows a single bit to be used to indicate that at least one child is in alarm or not ready for an interlock or permissive on a parent object. These would correlate to Bus[x].obj.CMDLLHMask.0 and Bus[x].obj.CMDLLHMask.8 respectively. CMDLLHMask is a temporary placeholder for children only status. The Sts_NrdyChildAlm output can be used with the Area, Unit, Equipment Module, and Equipment phase objects for this purpose.

Bus Faceplates

Many of the statuses and commands that are defined above can be found on the Bus Faceplate display. You can access the bus faceplate from the TreeView display or from object faceplates with the bus faceplate navigation button.

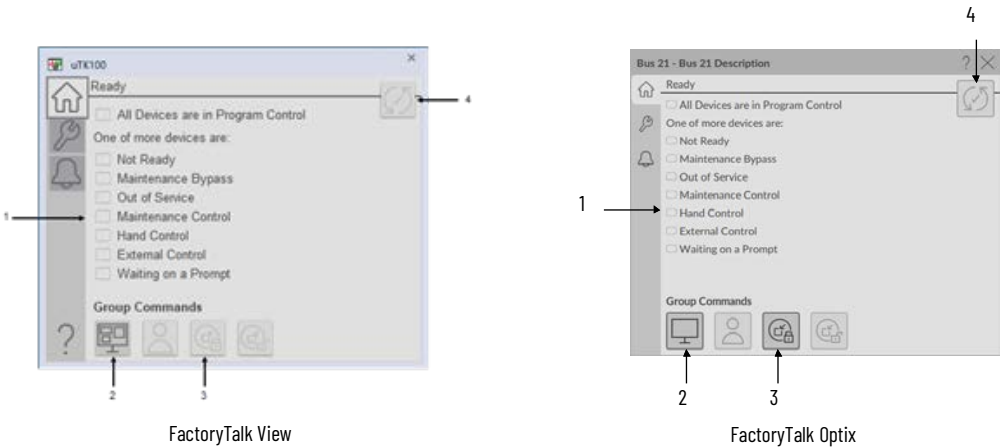


FactoryTalk View



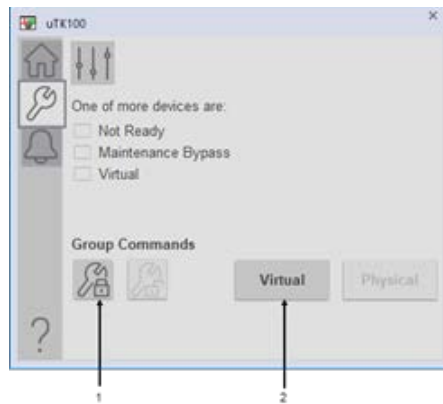
FactoryTalk Optix

Operator Tab

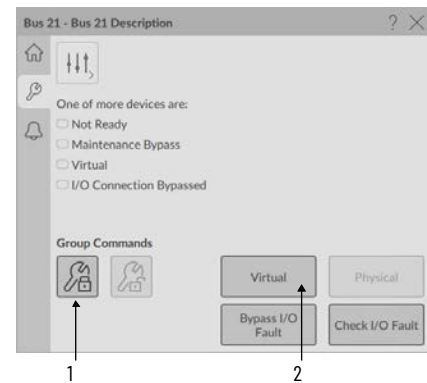


Item	Description
1	Child Status indication
2	Request Children Program or Operator Command Source
3	Acquire/Release Children External Command Source
4	Reset/Acknowledge All Child Alarms

Maintenance Tab



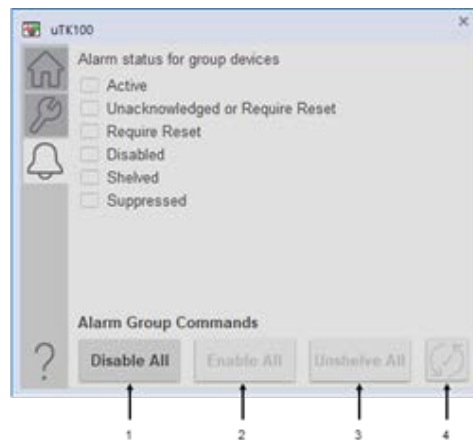
FactoryTalk View



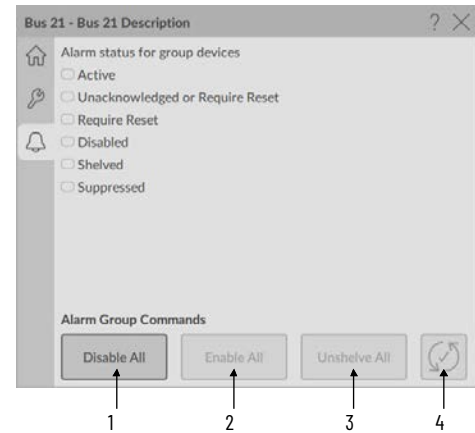
FactoryTalk Optix

Item	Description
1	Acquire/Release Children Maintenance Command Source
2	Request Children Virtual or Physical Mode

Alarms Tab



FactoryTalk View



FactoryTalk Optix

Item	Description
1	Disable all Child Alarms
2	Enable all Child Alarms
3	Un-shelve All Child Alarms
4	Reset/Acknowledge All Child Alarms

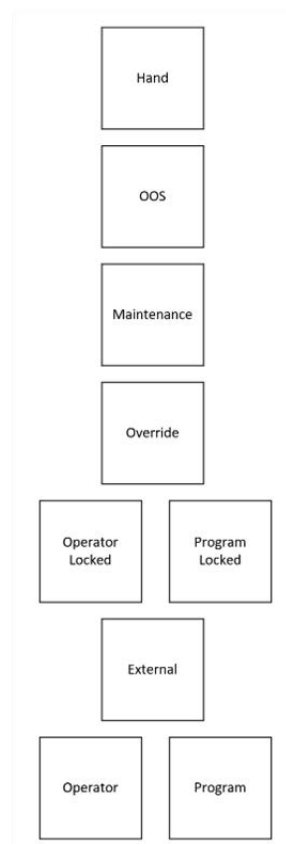
Bus Ownership

Ownership is built into the organizational bus backbone. It provides the ability for a parent in the tree to know that it has the child entity of interest assigned to it for control and that the child is placed in the program command source state for control by the parent. While the `raP_Opr_Owner` ownership function is available for use as a standalone component, the optional Add-On bus functionality within the process objects utilizes ownership based on the parent-child relationships that the user defines within an organizational tree. This modular ownership approach allows functional groups to be created by the control designer while the

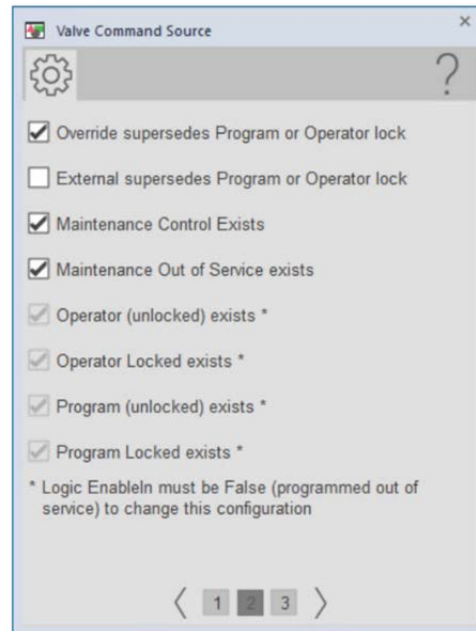
ownership rules are maintained independently in each process object command source configuration. The ownership and organization functions are asynchronous, which allows the designer to create an equipment module in code independent of the devices and equipment, which it uses and takes ownership of.

Ownership is an independent function from the command and status propagation that OOAP provides. While users can utilize the bus faceplate or bus input commands at a parent object to command a set of children to program mode, that method does not prevent one of those child objects from being taken out of program mode nor does it provide a means to check that a child is no longer controllable by program logic. By requesting ownership via the bus, the user can ensure that a set of children is going to be controlled by only that parent object's program logic. When a child object accepts a parent object ownership request, it is using that parent object's bus ID as the ownership ID. The parent object will then continuously reassert ownership of the program command source to that object, which locks the child into program mode. The raP_Opr_OrgScan Add-On Instruction continuously checks that all children of that parent are in the appropriate state for control (Organized) and that the child owner IDs match the bus index of that parent (Owned). This eliminates the need for large amounts of controller code that was previously required to manage and control groups of equipment.

When a child is owned by a parent via the bus, the program mode is the only command source class that is owned. The External, Override, Maintenance, OOS, and Hand command source states still maintain priority over the core program and operator command source classes regardless of ownership status. All underlying command source class prioritization rules remain the same regardless of if the object is owned or not. A child that is owned by a parent and is placed into the Program-Locked state can still be placed into Maintenance mode by a user that has sufficient security privileges. While the child is not in the program state, the parent of that child has a false children organized qualifier. When maintenance mode is released, the child returns to program locked and the parent regains the children organized status.



Bus ownership is dependent upon the underlying command source configuration of each child. For instance, if a child device program and program (locked) command source states are not configured to exist then bus ownership will not work. A child that is configured to have only the operator and program (locked) command source states to exist would return to operator mode each time ownership of that child is released or unbound.



The ability for a parent owner to operate the owned child control modules consists of two main qualifiers:

- Owned – The entity of interest has an owner ID of the interested owner assigned to it.
- Organized – The entity of interest is in the appropriate command source mode to be controlled.

The ownership function performs no action on the final control device, but instead provides a method to identify and prioritize owners so that other logic can use the 'Owned & Organized' status as permission for an Equipment Module or Equipment Phase to execute (Go / No Go Flag).

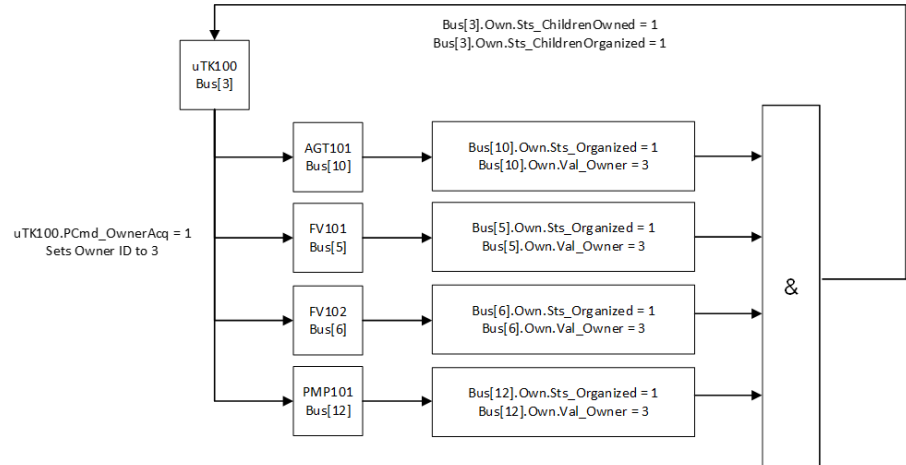
The parent bus element reports the status of its children with the aggregation of these two qualifiers. The parent process object Add-On Instructions (EMGen, EPGen, Sequencer, Area, and Unit) will automatically report these statuses. If the parent entity is not one of these objects, then the owned and organized statuses can be referenced at

Bus[Parent_Index].Own.Sts_ChildrenOwned and
Bus[Parent_Index].Own.Sts_ChildrenOrganized.

- Sts_ChildrenOwned – The aggregate of all children owned statuses. All children Val_Owner = Parent Bus Index.
- Sts_ChildrenOrganized – The aggregate of all children internal organized statuses. All children are in the program command source state.
- Sts_ChildrenGood – All children are owned and organized.

The following diagram illustrates the unit object uTK100 requesting ownership of four children using the uTK100.PCmd_OwnerAcq parameter. The raP_Opr_OrgScan Add-On Instruction monitors the owner ID and organized status of the children objects and defines a combined owned and organized status of the children for the unit object. The unit object then outputs

uTK100.Sts_ChildrenOwned and uTK100.Sts_ChildrenOrganized statuses to be referenced in logic as needed.



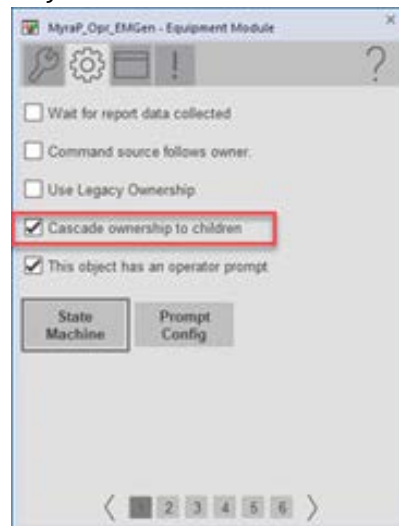
If a selected child object should not be owned when its parent requests ownership of its children, then the **raP_Opr_OrgDeviceCtrl** instruction can be used to issue a suppress ownership command at the child node. If ownership was suppressed at FV101 in the example above, then the **Bus[5].Own.Val_Owner** value would remain 0 while the **Bus[3].Sts_ChildrenOwned** and **Bus[3].Sts_ChildrenOrganized** values at the parent would remain equal to 1. The suppress ownership command leaves the FV101 object available to be owned by another parent in the organization while also allowing the parent object to proceed with its operation. For more information regarding the **raP_Opr_OrgDeviceCtrl** instruction, refer to [Chapter 7](#). This instruction is available in version 5.30.00 of the Process Library and later.

Ownership Configuration

The Sequencer, Area, Unit, Equipment Module, and Equipment Phase objects are designed to act as an owner (parent) in the organization. The implementation of Area, Unit, Equipment Module, Equipment Phase, and Sequencer is specific to each application. The tank farm example shows an instance of each type excluding the sequencer. These objects provide ownership configuration parameters that define how ownership can be requested from the object and how the resulting ownership statuses are managed.

Parent	Guidelines
Area	<p>An Area is the top-level ownership object.</p> <p>An Area groups Units together so that it can manage:</p> <ul style="list-style-type: none"> • Command Source for a group of equipment • Alarms for a group of equipment <p>If you must sequence equipment (rather than group equipment), use either a Sequencer, Equipment Module, or Equipment Phase</p>
Unit	<p>A Unit groups equipment. Units operate relatively independent of one another.</p> <p>A Unit manages:</p> <ul style="list-style-type: none"> • Command Source for a group of equipment • Alarms for a group of equipment <p>If you must sequence equipment (rather than group equipment), use either a Sequencer, Equipment Module, or Equipment Phase</p>
Equipment Module	<p>An Equipment Module groups and sequences equipment. Use an Equipment Module when you want to apply a custom state model to the equipment. For more information see EM/EP User Manual, publication PROCES-UM110.</p>
Equipment Phase	<p>An Equipment Phase groups and sequences equipment. Use an Equipment Phase when you apply the ISA 88.01 state model using PhaseManager™. For more information see PhaseManager Software User Manual, publication LOGIX-UM001. For more information see EM/EP User Manual, publication PROCES-UM110.</p>
Sequencer	<p>A sequencer groups and sequences equipment. Use a sequencer when you want to implement a procedure to operate equipment in a prescribed order. For more information see the Sequencer Object User Manual, publication PROCES-RM202.</p>

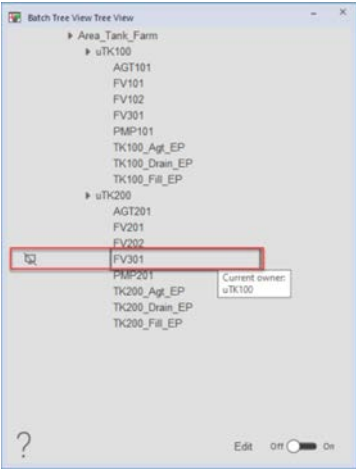
There can be many layers of parent/child relationships in an organizational tree. By default, the parent objects will automatically request ownership of its children when it becomes owned by its parent above it. The Sequencer, EMGen, EPGen, Area, and Unit objects can disable this functionality by setting the Cfg_CascadeOwn = 0.



The parental objects provide a Sts_NrdyChildNotUsable indication. This status indicates that the object requested ownership of its children and has not yet received an Owned and Organized status from at least one child. The annunciation of this status can be delayed with the Cfg_Child2BGoodTmr setting that is shown in the following display.



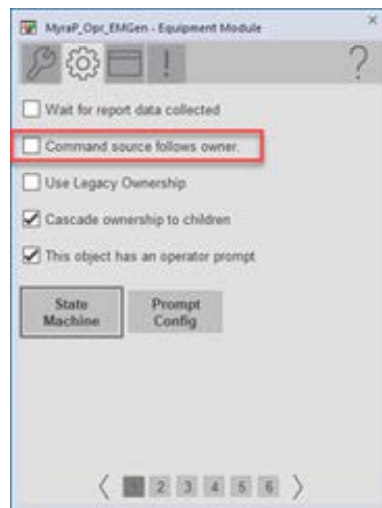
In the following tree display, FV301 is a shared device of both uTK100 and uTK200. Because FV301 is owned by uTK100 the FV301 node underneath uTK200 indicates a child not usable indication.



You can define which command source is able to request ownership of children. This is available in the EMGen, EPGen, Area, and Unit objects. This configuration can be used to prevent operators from requesting child ownership from the object faceplates. When Owner Commands is set to "Only Prog" or "Only Ext" then the faceplate buttons to Acquire and Release ownership will be disabled.



The "command source follows owner" configuration in the EMGen, EPGen, Area, and Unit objects can be used to automatically take ownership of children when the object is placed into the program command source state. Using this function disables operator ownership request functionality. If this configuration is used and the owner commands are set to only program in the command source exceptions faceplate then the children of the object will always be owned. This is not recommended if there are shared devices among parents in the system.



Although many objects (including PAI and PDI) can be included in the organizational tree, only objects with a Command Source can be owned. Objects are in Program command source state when owned by a parent Bus element.

The parent control point can issue ownership requests from the Operator, Program, External, or Maintenance command source states. All ownership requests and statuses propagate through the bus. Once owned, the owner function issues a request to place its children into the program state.

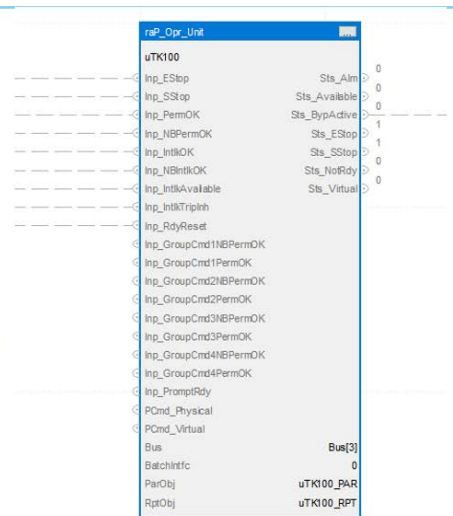
Ownership is established when a parent requests ownership, all its children grant this request, and all its children are placed into the program command source state (Children that belong to this parent are ready to accept parental requests). When a child grants a parent ownership request then the Owner ID of that child will be equal to the parent bus index.

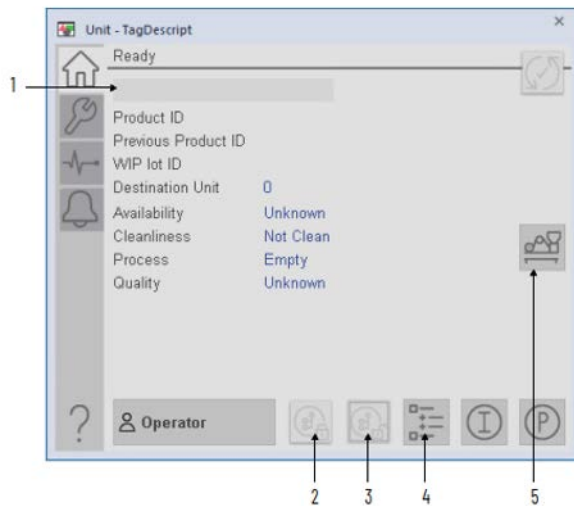
For example, a Unit object that is a parent can request ownership programmatically via the PCmd parameters. To request ownership, you must set PCmd_OwnerAcq = 1. The PCmd_OwnerRel parameter releases ownership. The command source exceptions for owner commands must be set to "Only Prog" or the unit must be in the program command source state to accept the PCmd_OwnerAcq and PCmd_OwnerRel when the command source exception is set to "Follow Source".

nsd - Data Type Properties - ns30301 (1, 62)

Parameters Tab

#	Id	Name	Is Required	Value	Data Type	Description
1		PCMD_Command			BOOL	Program command to acquire child command source
2		PCMD_Command2			BOOL	Program command to release child command source
3		PCMD_Command3			BOOL	Program command to update child command source
4		PCMD_Over			BOOL	Program command to select Program (Program to Control). The 1st
5		PCMD_Over2			BOOL	Program command to select Program (Program to Control). The 2nd
6		PCMD_Over3			BOOL	Program command to select Program (Program to Control). The 3rd
7		PCMD_Link			BOOL	Program command to lock Program (Status Control). The status
8		PCMD_Unlink			BOOL	Program command to unlock Program (Status Control)
9		PCMD_Param			BOOL	Program command to select command source (Status Control)
10		PCMD_Param2			BOOL	Program command to capture input parameter
11		PCMD_Param3			BOOL	Program command to capture output parameter
12		PCMD_Param4			BOOL	Program command to release input parameter
13		PCMD_Param5			BOOL	Program command to capture input result
14		PCMD_Param6			BOOL	Program command to capture result
15		PCMD_Param7			BOOL	Program command to release input result
16		PCMD_Param8			BOOL	External command to release memory (Program Over)
17		PCMD_Pri			BOOL	External command to release priority of PCMD_Command = 0
18		PCMD_Pri2			BOOL	External command to release PCMD_Command2
19		PCMD_Command4			BOOL	External command to release child command source
20		PCMD_Param9			BOOL	External command to capture input parameter
21		PCMD_Param10			BOOL	External command to capture output parameter
22		PCMD_Param11			BOOL	External command to release input result
23		PCMD_Param12			BOOL	External command to release output result
24		PCMD_Param13			BOOL	External command to release input result
25		PCMD_Param14			BOOL	External command to release output result
26		PCMD_Param15			BOOL	External command to release input result
27		PCMD_Param16			BOOL	External command to release output result
28		PCMD_Param17			BOOL	External command to release input result
29		PCMD_Param18			BOOL	External command to release output result
30		PCMD_Param19			BOOL	External command to release input result
31		PCMD_Param20			BOOL	External command to release output result
32		PCMD_Param21			BOOL	External command to release input result
33		PCMD_Param22			BOOL	External command to release output result
34		PCMD_Param23			BOOL	External command to release input result
35		PCMD_Param24			BOOL	External command to release output result
36		PCMD_Param25			BOOL	External command to release input result
37		PCMD_Param26			BOOL	External command to release output result
38		PCMD_Param27			BOOL	External command to release input result
39		PCMD_Param28			BOOL	External command to release output result
40		PCMD_Param29			BOOL	External command to release input result
41		PCMD_Param30			BOOL	External command to release output result
42		PCMD_Param31			BOOL	External command to release input result
43		PCMD_Param32			BOOL	External command to release output result
44		PCMD_Param33			BOOL	External command to release input result
45		PCMD_Param34			BOOL	External command to release output result
46		PCMD_Param35			BOOL	External command to release input result
47		PCMD_Param36			BOOL	External command to release output result
48		PCMD_Param37			BOOL	External command to release input result
49		PCMD_Param38			BOOL	External command to release output result
50		PCMD_Param39			BOOL	External command to release input result
51		PCMD_Param40			BOOL	External command to release output result
52		PCMD_Param41			BOOL	External command to release input result
53		PCMD_Param42			BOOL	External command to release output result
54		PCMD_Param43			BOOL	External command to release input result
55		PCMD_Param44			BOOL	External command to release output result
56		PCMD_Param45			BOOL	External command to release input result
57		PCMD_Param46			BOOL	External command to release output result
58		PCMD_Param47			BOOL	External command to release input result
59		PCMD_Param48			BOOL	External command to release output result
60		PCMD_Param49			BOOL	External command to release input result
61		PCMD_Param50			BOOL	External command to release output result
62		PCMD_Param51			BOOL	External command to release input result
63		PCMD_Param52			BOOL	External command to release output result
64		PCMD_Param53			BOOL	External command to release input result
65		PCMD_Param54			BOOL	External command to release output result
66		PCMD_Param55			BOOL	External command





Item	Description
1	Displays the current state of the object
2	Acquire child command source
3	Release child command source
4	Navigates to the tree view for this object
5	Navigates to the Bus faceplate

For object and visualization parameters, see Object and Visualization Parameters, publication [PROCES-RD201](#).

Additional commands exist on the bus data structure that pertain to ownership. The commands in the table below are used within parental object Add-On Instructions Sequencer, raP_Opr_EMGen, raP_Opr_EPGen, raP_Opr_Seq, raP_Opr_Unit, and raP_Opr_Area.

Parent bus ownership Commands	Bus Ownership Command Description
Cfg_CascadeOwn	Suppress automatic child ownership when a parent is owned by another parent. 1 = When this bus element is owned, do not automatically own its children.
PCmd_OwnerAcq/PCmd_OwnerRel	Parent request to own its children. 1 = Own children even if this element is not owned.
PCmd_UnbindChildren	Parent request to unbind all children. 1 = Allow ownership but do not enforce command source state. You are able to change the command source of the child while it is owned.

Child-level commands are available on the bus. These can be used in specific use cases where all children should not be affected by ownership the same way.

Child Bus Ownership Commands	Bus Ownership Command Description
Bus[x].Own.Inp_InitializeReq	Command to reinitialize the ownership of object x.
Bus[x].Inp_ProgDoNotInclude ⁽¹⁾	Input to unbind a bus object and not to include the object in the aggregation of Sts_ChildrenOrganized.

(1) Available in versions 5.10.01 and later.

Unbinding Ownership

When a parent acquires ownership of its children the parent will continuously reassert program class ownership over those children. By doing this, the owning parent is forcing the child command source state to Program. If the child object is configured to have the Program Lock and Program command source states, then it is forced to Program Lock. A Pcmd_Unlock command would only work for one scan of the controller before the parent reasserts ownership again and places the object back into Program Lock. Similar behavior would be exhibited if that object was configured to have only the program command source state. A user could attempt to place the object into the operator state from the object command source faceplate, but the parent would once again reassert ownership and place the device back into program on the next scan of the raP_Opr_OrgScan Add-On Instruction.

When there is a requirement to allow certain devices to change to operator mode while owned, the developer can utilize the unbind commands that are available on the bus or from the parental objects. An unbound child object will not continuously be forced to the program state by its parent. This allows a user to change the command source state of a child object to operator while still owned. A child object that is configured to have the Program Locked command source state transitions from Program Locked to Program mode when owned by a parent and an unbind command is present. Parental Add-On Instruction objects provide a Pcmd_UnbindChildren command to unbind all children of that parent while owned. If this command is used at a parent and one of the children are placed into operator mode, then the parent Add-On Instruction loses the children organized status and then loses the children good status. Unbind commands have no effect on the ownership of a child object. Parent Add-On Instruction objects output a Sts_ChildrenUnbound when the Pcmd_UnbindChildren command is used.

The Bus[index].Own.Inp_UnbindAll input can be used to selectively unbind all four ownership command sources simultaneously at any single process object that is connected to the bus. This may be desirable in cases where all children of an object may not need to be unbound. When using the Inp_UnbindAll input, the parent recognizes if it has lost organization when that child is placed outside of program mode.

Exclusion

Excluding an object from organized status accumulation at a parent unbinds that object and allows the operator to take it out of program mode without the parent object recognizing that the child is unorganized. Users can use the Bus[index].Inp_ProgDoNotInclude command to unbind and exclude a given bus object everywhere it is defined in the organizational tree. If an object has multiple parents in the tree and exclusion is only desired under a single parent, then the raP_Opr_OrgDeviceCtrl instruction can be used to unbind and exclude an object at a single node in the tree. For more information regarding the raP_Opr_OrgDeviceCtrl instruction, see to [Chapter 7](#). This instruction is available in version 5.30.00 of the Process Library and later.

The Bus[index].Inp_ProgDoNotInclude input can be utilized to selectively unbind individual process objects that are connected to the bus and exclude that object from the aggregation of the children organized status of its parent when that parent has requested ownership. This allows you to place that excluded object into operator mode while owned and still maintain the children organized and children good statuses at the parent object. This may be necessary in cases where the parent object is configured to stop or hold when its children are no longer organized. While an object is excluded from organized status, the object is still owned by the owning entity. The Bus[index].Inp_ProgDoNotInclude command should be set each time ownership is requested and reset at the time ownership is released. If unbind commands are maintained after ownership is released, then the process object will remain in its current command source state the next time ownership is requested. An example use case for the Inp_ProgDoNotInclude input would be for a PVSD or PPID child object that an operator may need to keep running while adjusting setpoint parameters manually. This input would allow the operator to place the object into operator mode to adjust the process without stopping or holding the parent object due to organization being lost.

Workflow

Before you begin, confirm that you have the following:

- Controller project and HMI project communicating to controller
- System hierarchy for equipment grouping
- Graphic framework installed
- Configured HMI shortcuts
- Displays created

IMPORTANT

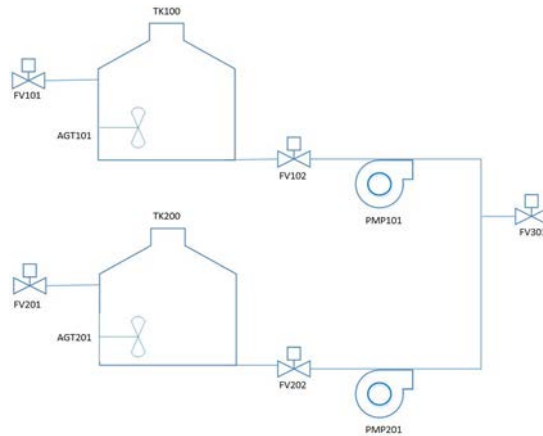
Application Code Manager and the PlantPAx Configuration Tool provide simplified workflows that are more efficient to initially configure organization. However, the manual process that is detailed in this document can be useful for minor edits.

The recommended workflow for large implementations of the bus is as follows:

1. Build the base project structure and device logic using Application Code Manager. The Process Library download provides ACM libraries that can define the controller code for the OrgScan and OrgView implementations. ACM provides interfaces to assign devices to bus elements in bulk. Object parameters can be set to automatically create mappings to the Bus[x].Own.Out_OwnerCmd and Bus[x].Own.Inp_OwnerSts as well as create an ArbitrationQ object where needed.
 2. Generate the .ACD file in Application Code Manager.
 3. Import the .ACD file into the PlantPAx Configuration Tool. Access the Organization Bus Configuration tool by right-clicking the imported controller and selecting "Configured Organization Bus."
 4. Configure bus element names and navigation from the organization bus configuration tool. All bus elements names can be set to the associated Logix tag names with a single click.
 5. Using the organization bus tool, drag-and-drop controller tags from the logical organizer window to the BusNode Tree window to easily define the parent and child relationships in the organization.
 6. Configure the bus and node element command and status propagation masking.
 7. Configure the FactoryTalk® View SE startup macro definitions and graphic displays.
-

Manually Configure Organizations

1. In a process controller project, create the organization logic (create Bus and Node arrays)
2. Define the Bus elements
3. Implement the OrgView elements that store data for each HMI client
4. Create the Organizational Tree (define nodes and configure propagation and displays)



The Tank Farm Area has:

- Unit TK100 has children FV101, FV102, FV301, PMP101, and AGT101
- Unit TK200 has children FV201, FV202, FV301, PMP201, and AGT201

Create the Organization Logic

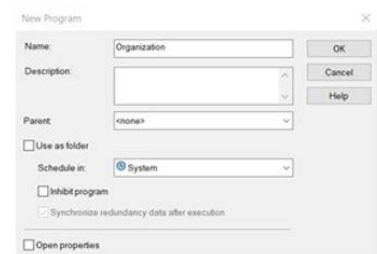
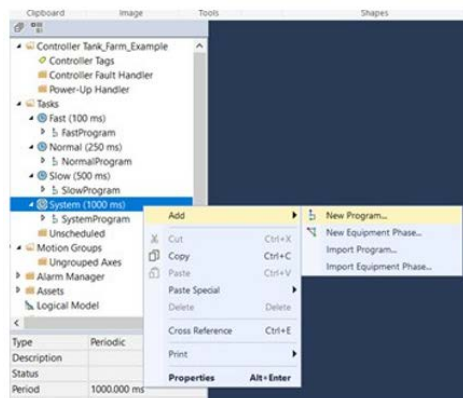
Before you begin, import the process library Add-On Instructions into your controller project. All logic must be in one controller. The following Add-On Instructions are required:

- raP_Opr_OrgScan Organizational Scan
- raP_Opr_OrgView Organizational View

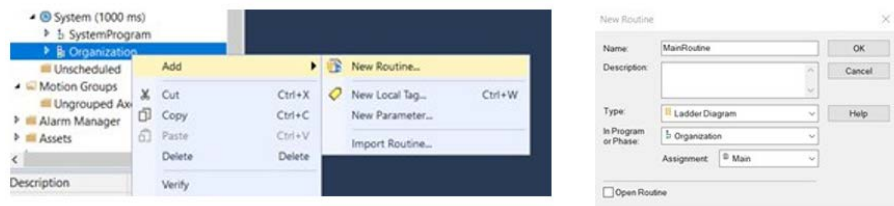
Create the Organization Program

In a process controller project, create the organization program.

1. Add a new program in the appropriate Task and name it Organization. The default task is the System task.



2. Add a new routine to the Organization program and name it MainRoutine.



3. Open the MainRoutine in the Ladder Diagram editor and add an raP_Opr_OrgScan Add-On Instruction to manage the propagation of commands/status and Ownership within the Organization Tree.



Create Array Tags

Create two array tags for these parameters that are used in this Add-On Instruction:

- Bus array with an element for each device in your system hierarchy. Make the array larger than the number of devices so you have room for future additions.
 - BusNode array with an element for each device and its relationship to other devices. Make the node array twice as large as the Bus array so that you can account for devices that have relationships with multiple other devices, and you have room for future additions
1. Create controller-scoped tag OrgScan of data type raP_Opr_OrgScan.



2. Create a controller-scoped array tag Node of type raP_UDT_Opr_Bus_Node.

The array tag must be named Node. For this example, edit the array to have 100 elements. You can have more nodes than bus elements to define multiple parent/child relationships.

The screenshot shows the 'New Parameter or Tag' dialog box. The 'Name' field is set to 'Node'. The 'Usage' is 'Local Tag'. The 'Type' is 'Base'. The 'Data Type' is 'raP_UDT_Opr_Bus_Node[100]'. The 'Scope' is 'Organization'. The 'External Access' is 'ReadWrite'. The 'OPC UA Access' is 'None'. The 'Style' is 'Constant'. The 'Sequencing' checkbox is unchecked. The 'Open Configuration' and 'Open Parameter Connections' checkboxes are also unchecked.

3. Create a controller-scoped array tag Bus of type raP_UDT_Opr_Bus.

The array tag must be named Bus. Edit the array elements to have 50. Create more elements than your original list of bus elements to leave space to add future elements.

The screenshot shows the 'New Parameter or Tag' dialog box. The 'Name' field is set to 'Bus'. The 'Usage' is 'Local Tag'. The 'Type' is 'Base'. The 'Data Type' is 'raP_UDT_Opr_Bus[50]'. The 'Scope' is 'Organization'. The 'External Access' is 'ReadWrite'. The 'OPC UA Access' is 'None'. The 'Style' is 'Constant'. The 'Sequencing' checkbox is unchecked. The 'Open Configuration' and 'Open Parameter Connections' checkboxes are also unchecked.

IMPORTANT You can have as many as 500 Bus elements per controller (1500 for 5.00.04 or later)

4. Add a rung with an instance of the raP_Opr_OrgView Add-On Instruction for each HMI client that will be using the organizational tree. This example assumes five HMI clients, so there are five individual rungs.

Right-click the raP_Opr_OrgView ? and create a tag OrgView of data type raP_Opr_OrgView.

The screenshot shows the 'raP_Opr_OrgView' instruction block. It has four inputs: 'raP_Opr_OrgView', 'BusNode', 'Bus', and 'OrgScan'. Each input has a question mark icon next to it, indicating that a tag needs to be created for each.

5. Edit the Data Type and enter the array dimensions so that you have an element for each HMI client. In this example there are five HMI clients so edit the array to have five elements (one element for each of the five HMI clients in this example).

IMPORTANT

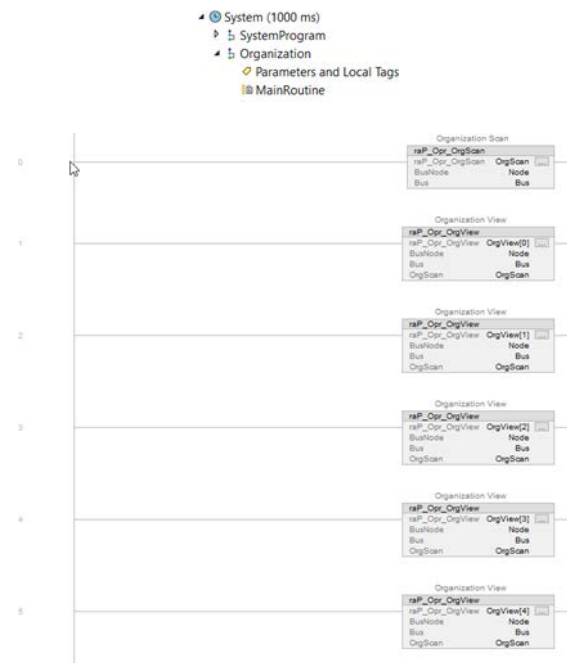
Each HMI Client startup macro must specify which of the five OrgView elements it is assigned to. Correlating an OrgView element to each unique client in the system allows for users to view and navigate the organizational tree view independently from each other. Users can also configure the Organization view to behave differently from client to client. Editing of the tree can be disabled for a particular client from the OrgView Config faceplate. Each OrgView element can maintain a different starting node for each client. When a start node is defined for a client, that starting node will be the uppermost node in the tree view in place of the <root> node.



6. Enter the BusNode, Bus, and Orgscan tags you created for the OrgScan instruction.
Select one OrgView[x] element for each rung/instance to correspond with each HMI client.

Example Logic

The final logic for this example looks like the following:

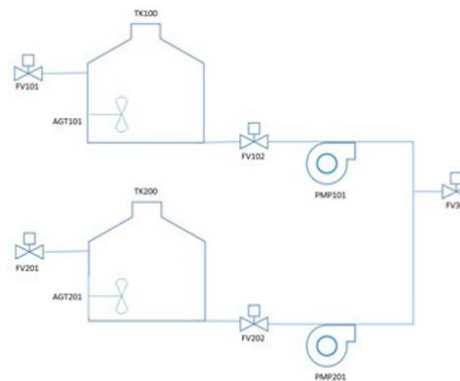


Define the Bus Elements

Organization requires a mechanism to collect entities into a structure. Each entity that is grouped

into an organization must be assigned a unique identifier. The Bus array provides this mechanism, in addition to providing an interface to exchange commands and status. Each PlantPAx bus-capable control element must be assigned a bus element. Generic bus elements can be added as logical containers as necessary to group control elements and other generic bus elements.

Example Bus Elements



You need a bus element for every possible owner (parent) and child. For this example, create these elements. The elements can be in any order, and you can use any names that are appropriate for your application.

Bus Element	Name	Description
Bus[0]		Reserved
Bus[1]		
Bus[2]	Area_Tank_Farm	Process area
Bus[3]	uTK100	Unit TK100
Bus[4]	uTK200	Unit TK200
Bus[5]	FV101	Valve FV101 owned by TK100
Bus[6]	FV102	Valve FV102 owned by TK100
Bus[7]	FV201	Valve FV201 owned by TK200
Bus[8]	FV202	Valve FV202 owned by TK200
Bus[9]	FV301	Valve FV301 shared by TK100 and TK200
Bus[10]	AGT101	Agitator AGT101
Bus[11]	AGT201	Agitator AGT201
Bus[12]	PMP101	Pump PMP101
Bus[13]	PMP201	Pump PMP201
Bus[14]	TK100_Fill_EP	Fill Equipment Phase for TK100
Bus[15]	TK100_Drain_EP	Drain Equipment Phase for TK100
Bus[16]	TK100_Agt_EP	Agitate Equipment Phase for TK100
Bus[17]	TK200_Fill_EP	Fill Equipment Phase for TK200
Bus[18]	TK200_Drain_EP	Drain Equipment Phase for TK200
Bus[19]	TK200_Agt_EP	Agitate Equipment Phase for TK200
Bus[20]...Bus[50]	Empty; for future additions	

IMPORTANT Release 5.00.00 of the PlantPAx library, supports as many as 500 bus elements. Release 5.00.04 and later supports as many as 1500 bus elements.

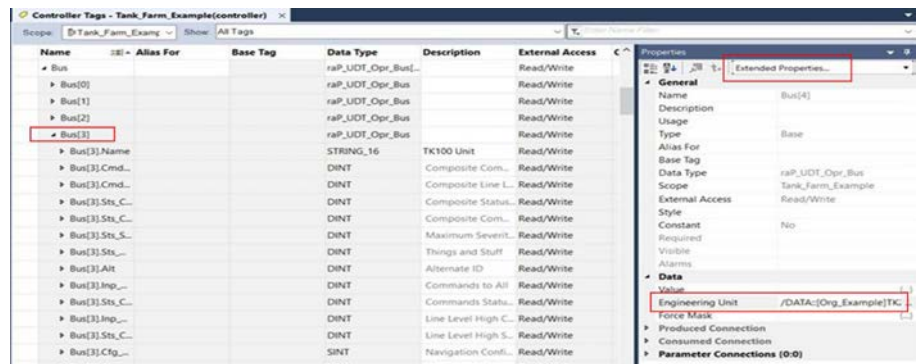
Each bus element has multiple members. To see the members, on the OrgScan or OrgView instruction, right-click Bus and select Monitor "Bus".

Expand each element to configure the device.

1. Enter the name of each element in the Value Field for each Bus[x].Name. The organizational tree uses this value for the device name.
2. Add Extended Properties > Engineering Unit to the element and specify the shortcut for the Data value. The shortcut ties the element to the display faceplate. The shortcut must be defined in the HMI project.

In this example:

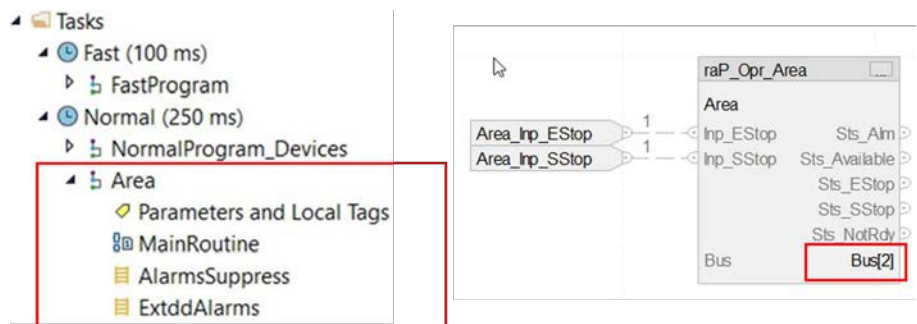
- DATA is the name of the data server
- Org_Example is the name of the shortcut in the HMI project
- uTK100 is the name of the element



Configure the Area Instance

Import the PlantPax area control strategy: CS_raP_Opr_Area. This defines the Area. In this example, the main routine identifies Bus[2] element for the area.

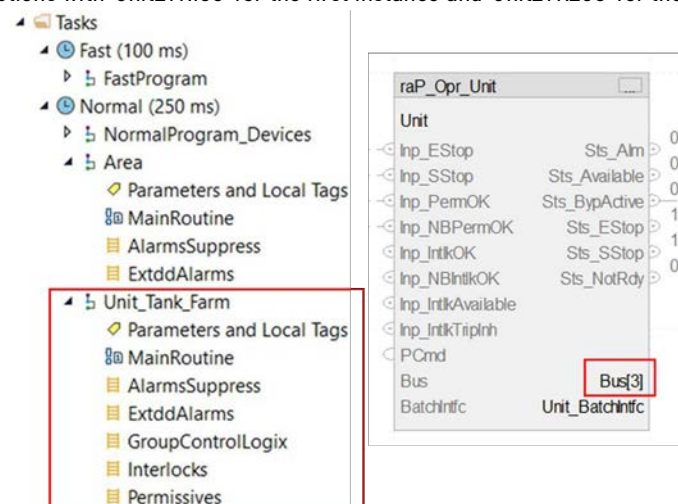
On the Import Configuration dialog, find and replace all instances of 'raP_Opr_Area' in Tags and Descriptions with 'Area.'



Configure the Unit Instances

Import the PlantPax unit control strategy: CS_raP_Opr_Unit. This defines a Unit. In this example, the main routine defines Bus[3] (Unit_TK100) and Bus[4] (Unit_TK200) elements for a Unit.

On the Import Configuration dialog, find and replace all instances of 'raP_Opr_Unit' in Tags and Descriptions with 'Unit_TK100' for the first instance and 'Unit_TK200' for the second instance.



Create instances for the TK100 and TK200 Units.

Configure the Equipment Phase or Equipment Module Instances

Import the appropriate PlantPAX control strategy:

- CS_raP_Opr_EM_Gen (Equipment Module)
- CS_raP_Opr_EP_Gen (Equipment Phase)

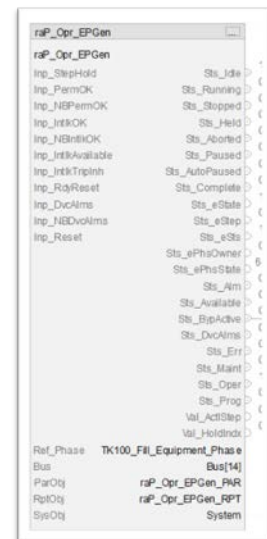
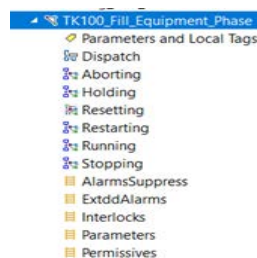
This example uses Equipment Phases. Edit the Bus element appropriately:

- Bus[14] for Unit TK100_Fill_EP
- Bus[15] for Unit TK100_Drain_EP
- Bus[16] for Unit TK100_AGT_EP
- Bus[17] for Unit TK200_Fill_EP
- Bus[18] for Unit TK200_Drain_EP
- Bus[19] for Unit TK200_AGT_EP

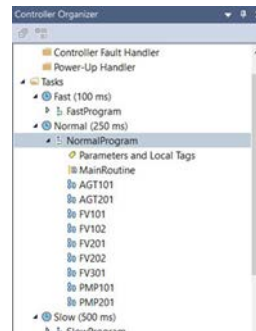
On the Import Configuration dialog, find and replace all instances of 'raP_Opr_EP_Gen' in Tags and Descriptions with the appropriate phase name.

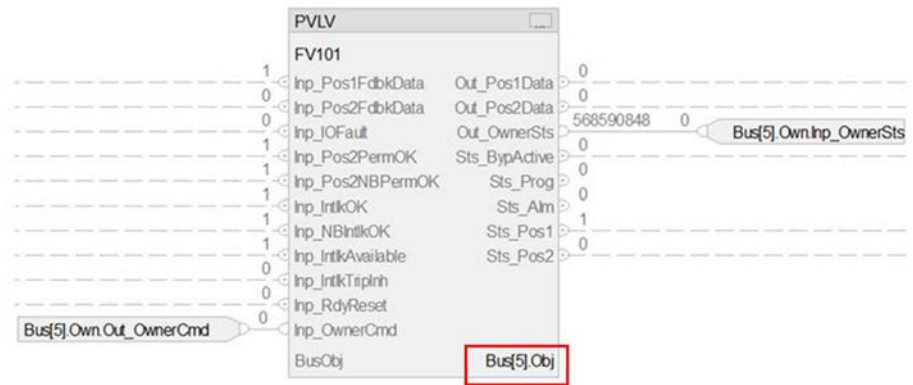
Configure the Device Instances to Use the Bus Elements

Use the PlantPAX control strategies to create the logic for your application. Import the appropriate control strategy for each device. Assign the bus element of each device to the process instruction.

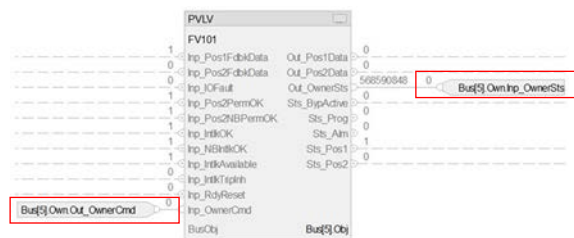


The Bus Object must match the device element in the Bus array.





Each device that you want to add to the node tree must have an associated process instruction.

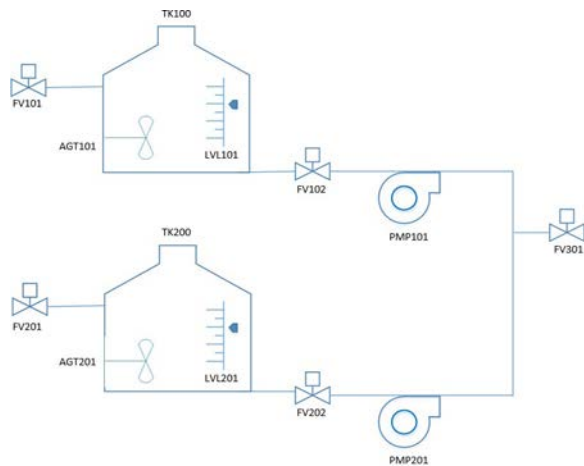


Each process object that requires a BusObj connection also requires the mapping of the Bus[x].Own.Out_OwnerCmd to the Object.Inp_OwnerCmd and the Bus[x].Own.Inp_OwnerSts to the Object.Out_OwnerSts. This mapping allows for bus ownership to place objects in the requested command source state and then report that status back to the parent object. The Bus[x].Own index references should match the Bus[x].Obj index reference. Bus capable Add-On Instructions do not require this mapping because the bus connection is for the entire Bus UDT structure. The Bus.Own mapping exists internal to the instruction in this case.

Add Devices

Bus elements can be in any order, and you can use any names that are appropriate for your application. To add additional devices to the Bus, reference and use a Bus array element that is not being used by another object, excluding 0.

In this example, add level indicators to both tanks.



Bus Element	Name	Description
Bus[20]	LVL101	Level for TK100
Bus[21]	LVL201	Level for TK200
Bus[22]...Bus[50]	Empty; for future additions	Bus[22]...Bus[50]

Define the OrgView Elements

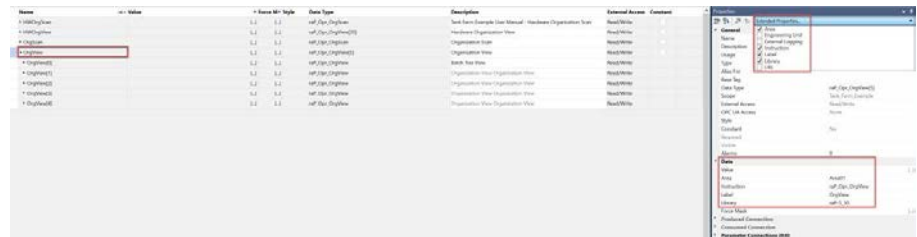
Each control entity and container has been assigned a unique identifier (Bus element), and each tree and its entities can be updated dynamically using the OrgScan instruction (Node element).

To facilitate the creation of the HMI client access, you must also define an OrgView array, which stores the data for each client. This is an array of type raP_UDT_Opr_OrgView. There should be one OrgView instruction that is configured for each HMI Client. OrgView provides a window to allow online manipulation of each organizational tree. The Organization View object allows online changes to the Organization Tree (Nodes) while providing a view to the various organizational trees. Only one client can edit the organization at a time. You must request edit control before making updates. A configurable timeout is used to release edit control at expiration.



Example OrgView Elements

For each OrgView element, enable the Area, Instruction, and Library Extended Properties. These properties tie the display back to the organization structure.

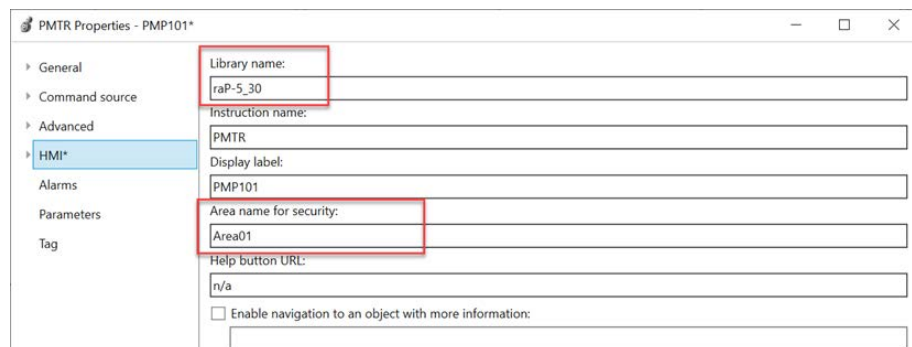


Then in the Data, enter this information:

- Area = Area01
- Instruction = raP_Opr_OrgView
- Library = raP-5_30
- URL = n/a

IMPORTANT

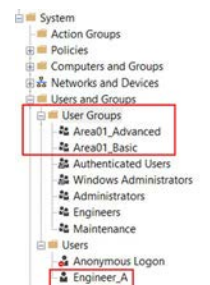
The Area and Library names must match the settings in the HMI page of the device properties. The library parameter must match the latest version of that library at the time of implementation. See the following example:



These settings are the default names when you import a process control strategy.

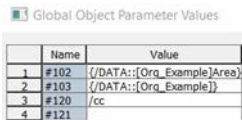
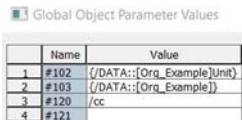
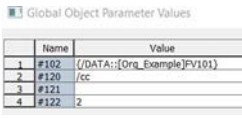
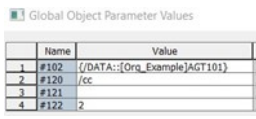
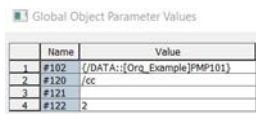
Create the Organizational Tree in the HMI Client

Make sure that you have the application configured with the appropriate user groups. This example uses the default Area01_Basic and Area01_Advanced user groups with a user Engineer_A.

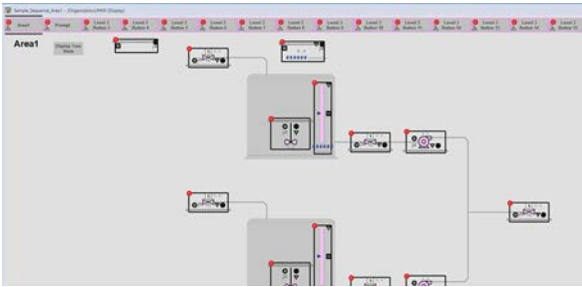


Configure the Client Display

1. Build the client display with the elements you defined in the Bus. This example uses these graphic symbols in the display:

Graphic Symbol	Description
Area (You must have one Area symbol)	(raP-5_30-SE) Graphic Symbols - raP_Opr_Area.ggfx 
Unit	(raP-5_30-SE) Graphic Symbols - raP_Opr_Unit.ggfx 
Valve (one solenoid symbol for each valve)	(raP-5_30-SE) Graphic Symbols - PVLV.ggfx 
Agitator (one agitator symbol for each agitator)	(raP-5_30-SE) Graphic Symbols - PMTR.ggfx 
Pump (one blower symbol for each pump)	(raP-5_30-SE) Graphic Symbols - PMTR.ggfx 
Display Tree View button	(raP-5_30-SE) Graphic Symbols - Cross Functional.ggfx

2. Select and copy the 'Display Tree View' button from the '(raP-5_30-SE) Graphic Symbols - Cross Functional.ggfx' global object file. Paste this button to the desired display.



3. Add the following lines to the client startup macro. The Template_ClientStartup macro file provides an example that can be uncommented and adjusted as needed. The template example uses the OrgView element 0 and /Area1/DATA::[Hardware] for the controller topic. Another startup macro must be created for each HMI client in the system so that each client can be assigned another OrgView element. The macros DefineShowTreeCmd and DefineShowHWTreeCmd are referenced in the startup macros and must exist in the project.

Line	Description
Define SW_RedefineShowTreeCmd DefineShowTreeCmd X	X = Client number (OrgView element) 0 in this example for client 0
SW_RedefineShowTreeCmd /Area/DATASERVER::[TOPIC]	Area = Area for data server DATASERVER = data server name TOPIC = shortcut name (path to controller) DATA::[Org_Example] in this example

See [Client Startup Macro on page 81](#) to complete this configuration.

IMPORTANT The PlantPax Graphic Framework Tool provides an interface for defining hardware and software bus commands for the client startup macros in the L1 configuration.

4. Generate the client display.

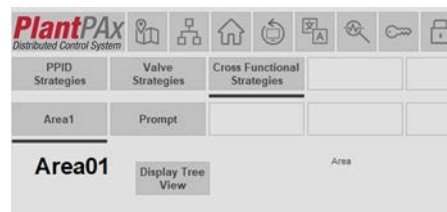


Build the Node Tree (FactoryTalk View)

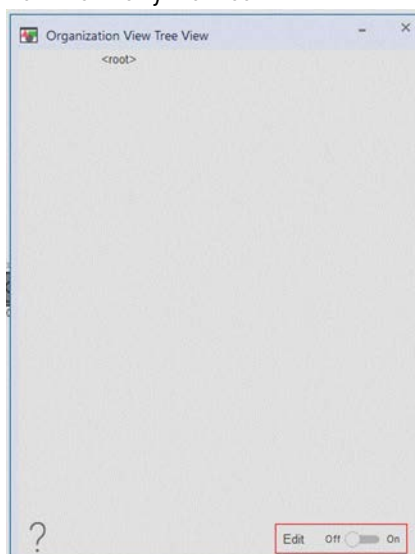
For FactoryTalk Optix See [Build the Node Tree \(FactoryTalk Optix\) on page 59](#).

The node tree is where you define system hierarchy and relationships among objects. Build the complete node tree once. You can specify a start node within the node tree or each client. These examples show how to manually perform the processes, you can also use the PlantPax Configuration Tool.

1. Sign in to the client display and click the Display Tree View button.



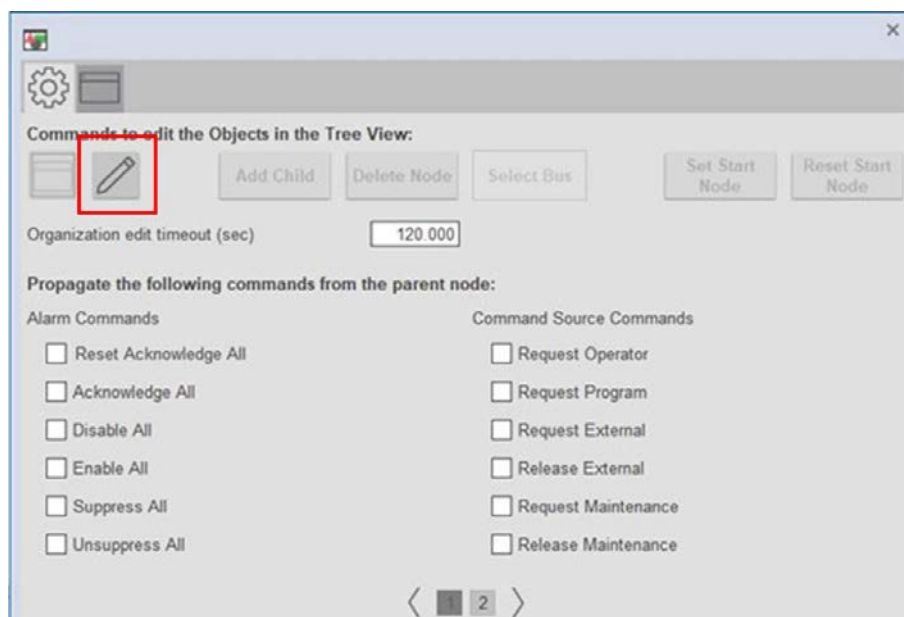
The initial Tree View shows only the <root>.



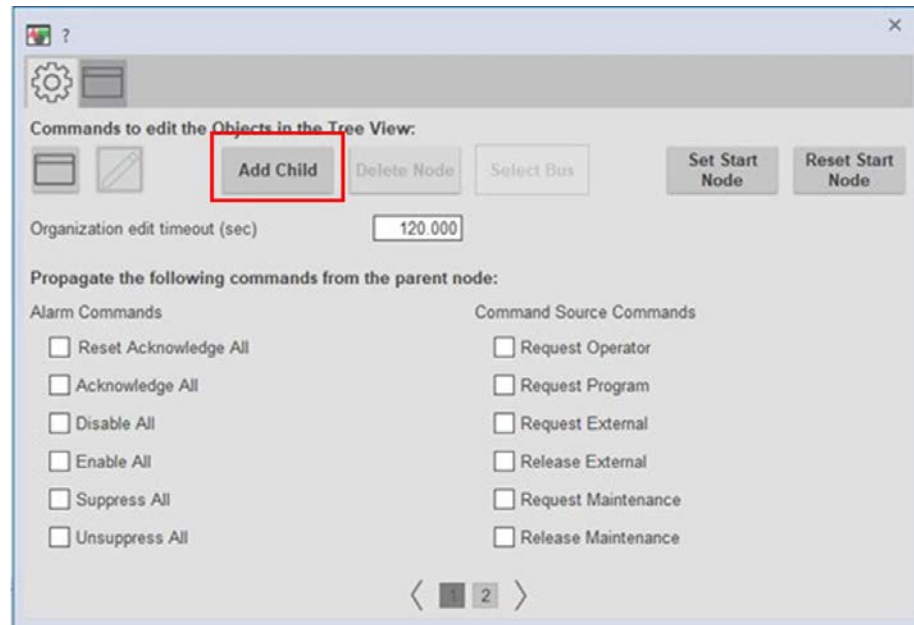
2. Enable Edit mode

In Edit Mode, you can define the Bus that is associated with each Node, define parents or children, and define propagation configuration.

3. Select <root> to display the edit options.
4. Select the edit button.



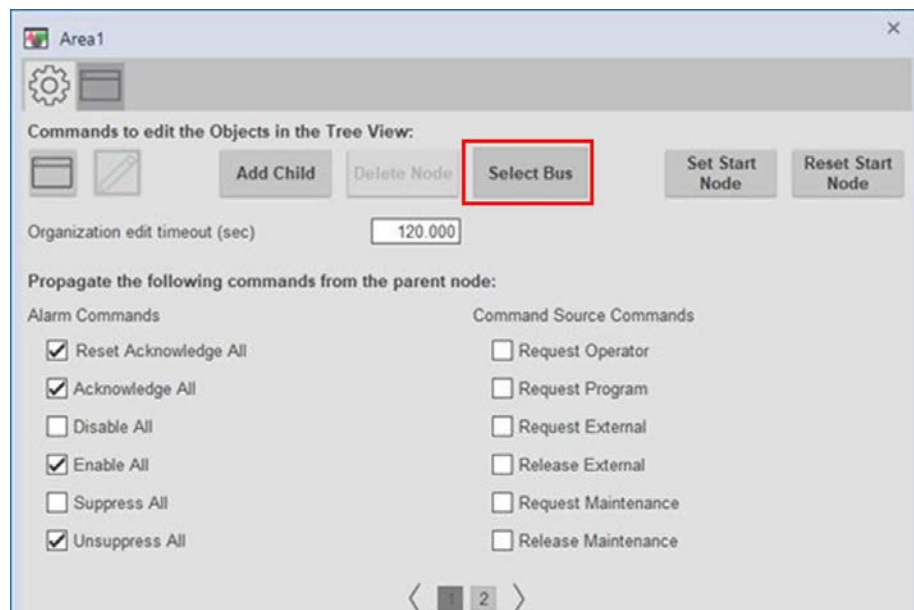
5. Select the Add Child button to add a node.



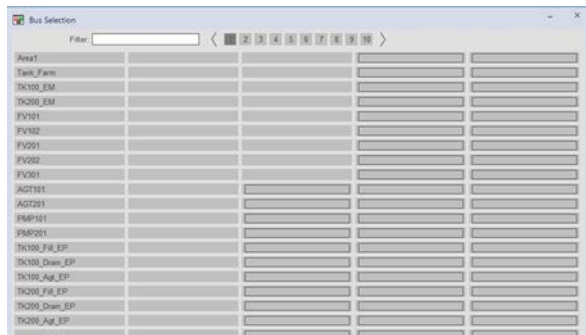
This adds a ? to the display tree.



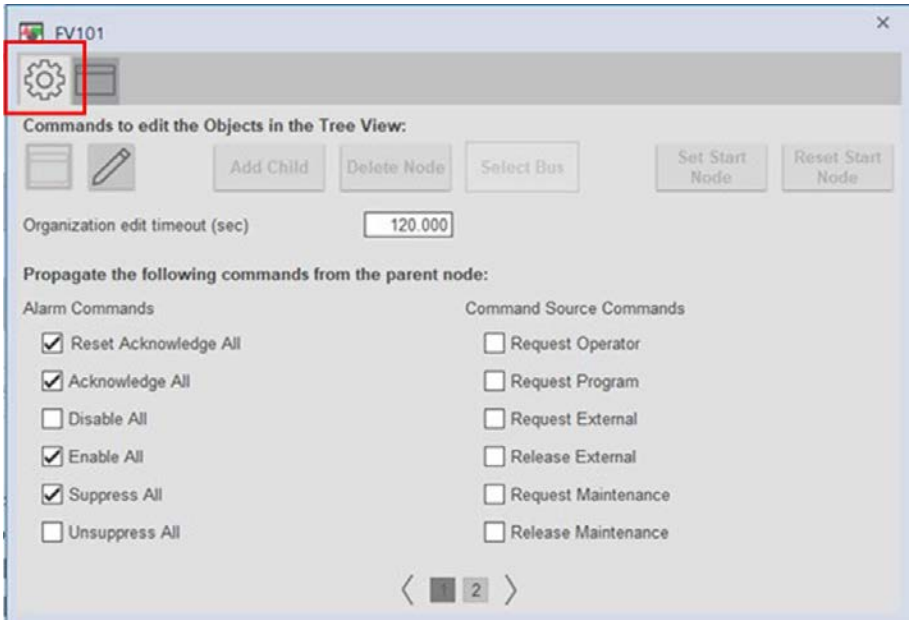
6. Select the ? to edit the node.
7. Select the edit button and click the Select Bus button to assign a bus element to the new node element.



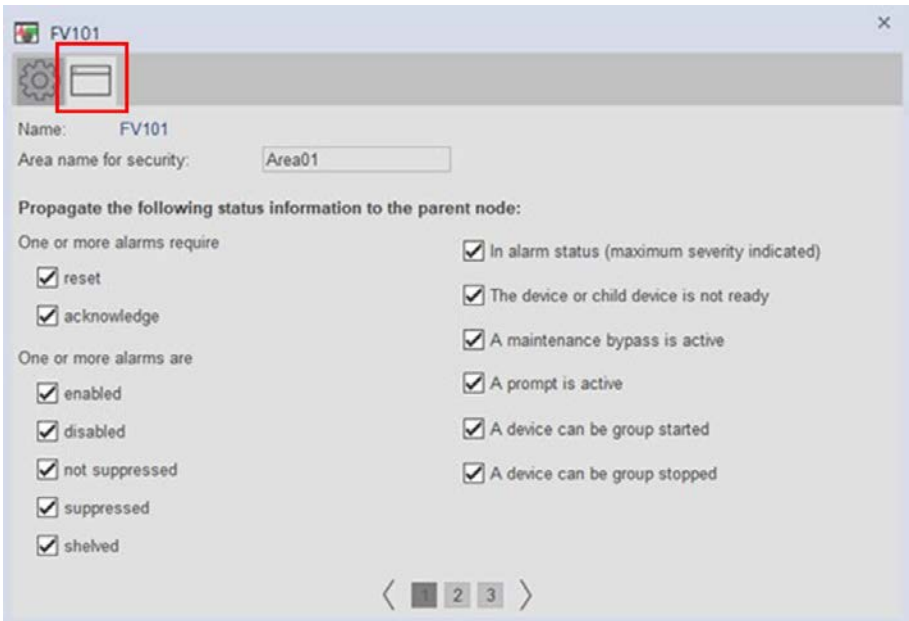
8. Select the bus element.



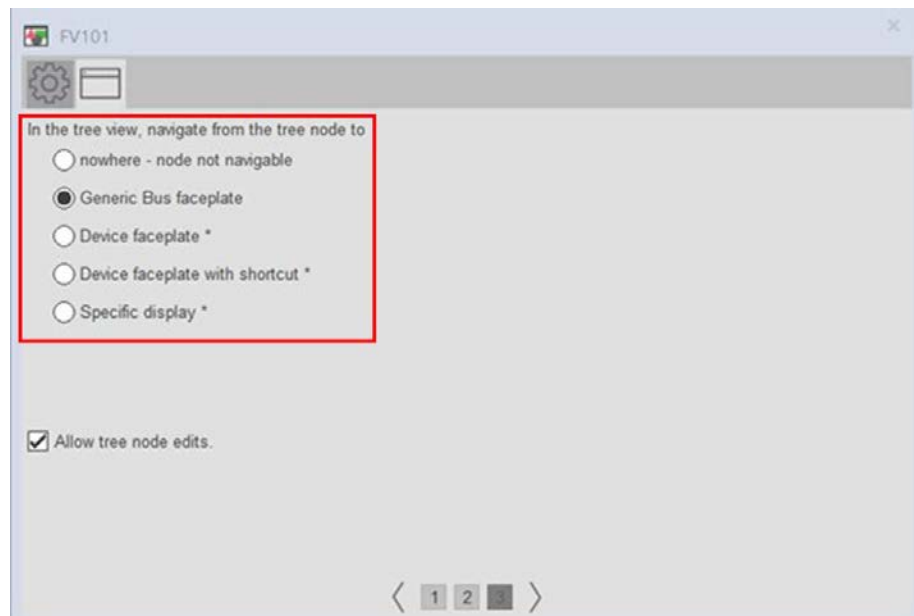
9. Define how to propagate commands (Engineering Tab).



10. Define how to propagate status (HMI Tab).



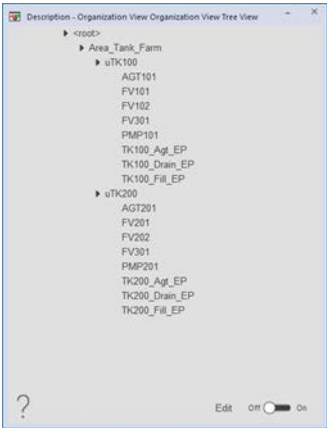
11. Define navigation (HMI Tab, page 3).



IMPORTANT Navigation selections above that are marked with an * use the Extended Tag Property @EngineeringUnit for the Bus Array instance (for example, Bus[37].@EngineeringUnit) because the @Navigation Extended Tag Property is not available in the user-defined data type (UDT) bus array.

Option	Description	Engineering Unit Extended Tag Property
nowhere - node not navigable	There is no configured navigation for the node	not used
Bus Faceplate	For use when the node does not have an associated display or faceplate. The generic faceplate can be configured to show any of the standard bus commands or status information.	not used
Device Faceplate	Show a standard library faceplate (or other faceplate that uses the same navigation method)	Device Tag. For example, "/DATA::[Tank_Farm]AGT101"
Device Faceplate with Shortcut	For use with faceplates that require the path as the third display parameter (These include Sequencer, Area, Unit, EM, and EP)	Device Tag. For example, "/DATA::[Tank_Farm]uTK100"
Specific Display	A custom display. Display parameters are: #1 - read/write tag (constant 2) #2 - Bus instance tag from the bus array #3 - Area security tag from OrgView	FactoryTalk View display name

Complete Organization Tree for this Example



The organization tree updates the Node array in the controller.

Node Element	Organization Tree Element
Node[0]	<root>
Node[1]	Area_Tank_Farm
Node[2]	uTK100
Node[3]	uTK200

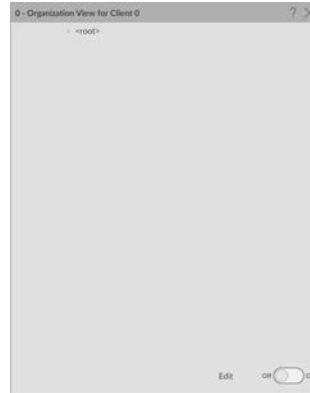
Name	Value	Force Mask	Style	Data Type	Description
Node		[...]	[...]	rsP_UDT_Opr_Bus_No...	
Node[0]		[...]	[...]	rsP_UDT_Opr_Bus_No...	
Node[1]		[...]	[...]	rsP_UDT_Opr_Bus_No...	
Node[2]		[...]	[...]	rsP_UDT_Opr_Bus_No...	
Node[3]		[...]	[...]	rsP_UDT_Opr_Bus_No...	
Node[3].BusIndex		4	Decimal	INT	Bus Array Index of A...
Node[3].ChildCnt		8	Decimal	INT	Number of Children
Node[3].FirstChildIndex		12	Decimal	INT	Index of First Child
Node[3].ParentIndex		1	Decimal	INT	Node Array Index of ...
Node[3].Sts		0	Decimal	INT	Things and Stuff
Node[3].Cfg_Ownership		0	Decimal	INT	Configuration for O...
Node[3].Cfg_StsMask	2#0000_0000_0000_0000_0000_0000_0000		Binary	DINT	Bits 0: Push Status to...
Node[3].Cfg_CmdMask	2#1111_1100_0000_0011_1111_1100_0111_1110		Binary	DINT	Bits 1: Accept Comm...
Node[3].Cfg_CmdLLHMask	2#0000_0000_0000_0000_0000_0011_0000_1111		Binary	DINT	Bits 1: Accept Comm...

Build the Node Tree (FactoryTalk Optix)

For FactoryTalk View See [Build the Node Tree \(FactoryTalk View\) on page 53](#).

The node tree is where you define system hierarchy and relationships among objects. Build the complete node tree once. You can specify a start node within the node tree or each client. These examples show how to manually perform the processes, you can also use the PlantPAx Configuration Tool.

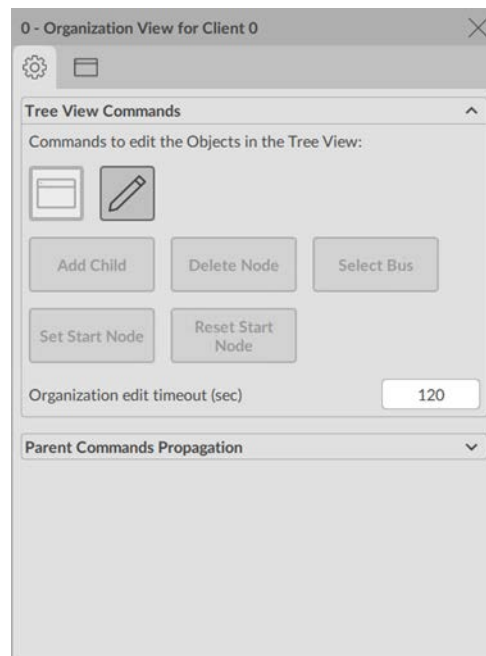
1. Click the Display Tree View button to open the initial Tree View.



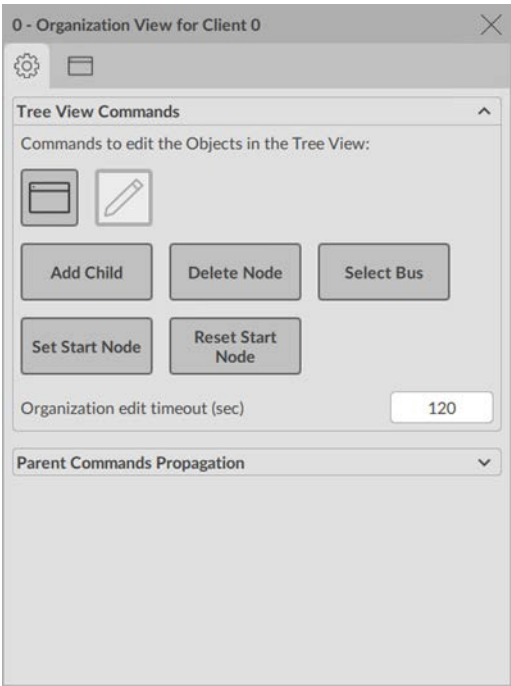
2. Enable Edit mode

In Edit Mode, you can define the Bus that is associated with each Node, define parents or children, and define propagation configuration.

3. Select <root> to display the edit options.
4. Select the edit button.

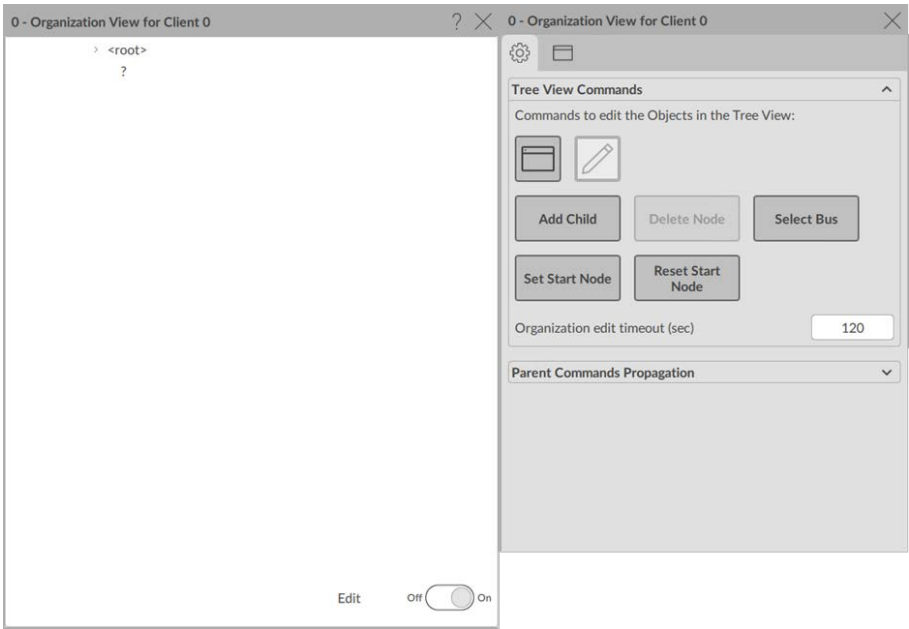


- 5. Select the Add Child button to add a node.

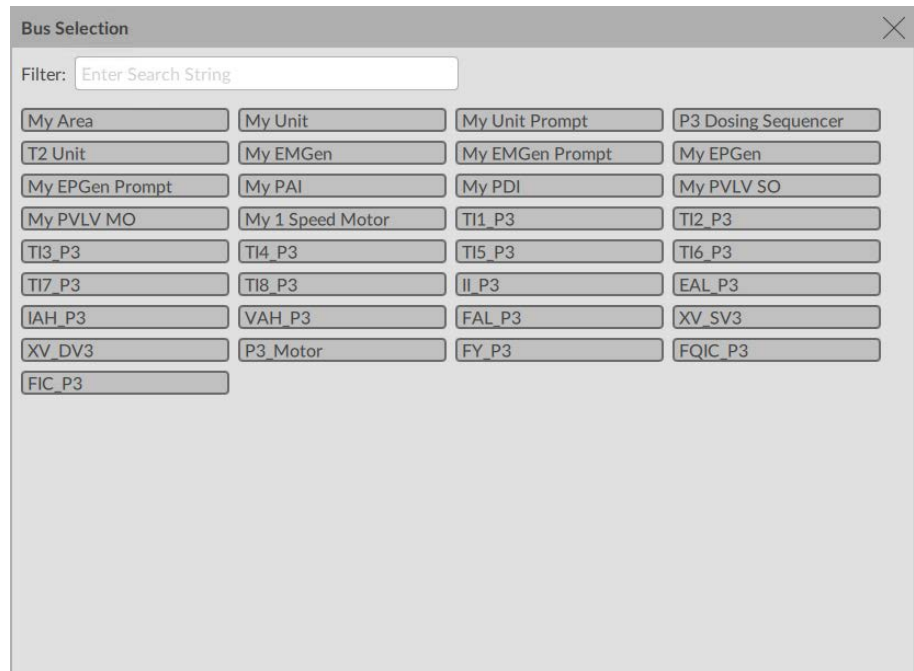


This adds a ? to the display tree.

- 6. Select the ? to edit the node.
- 7. Select the edit button and click the Select Bus button to assign a bus element to the new node element.



8. Select the bus element.

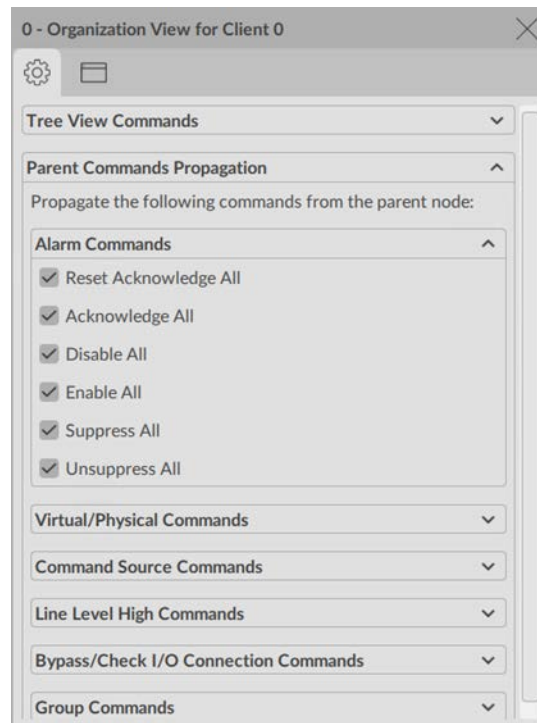


Bus Selection

Filter:

My Area	My Unit	My Unit Prompt	P3 Dosing Sequencer
T2 Unit	My EMGen	My EMGen Prompt	My EPGen
My EPGen Prompt	My PAI	My PDI	My PVLV SO
My PVLV MO	My 1 Speed Motor	TI1_P3	TI2_P3
TI3_P3	TI4_P3	TI5_P3	TI6_P3
TI7_P3	TI8_P3	II_P3	EAL_P3
IAH_P3	VAH_P3	FAL_P3	XV_SV3
XV_DV3	P3_Motor	FY_P3	FQIC_P3
FIC_P3			

9. Define how to propagate commands (Advanced Engineering Tab).



0 - Organization View for Client 0

Tree View Commands

Parent Commands Propagation

Propagate the following commands from the parent node:

Alarm Commands

- ☒ Reset Acknowledge All
- ☒ Acknowledge All
- ☒ Disable All
- ☒ Enable All
- ☒ Suppress All
- ☒ Unsuppress All

Virtual/Physical Commands

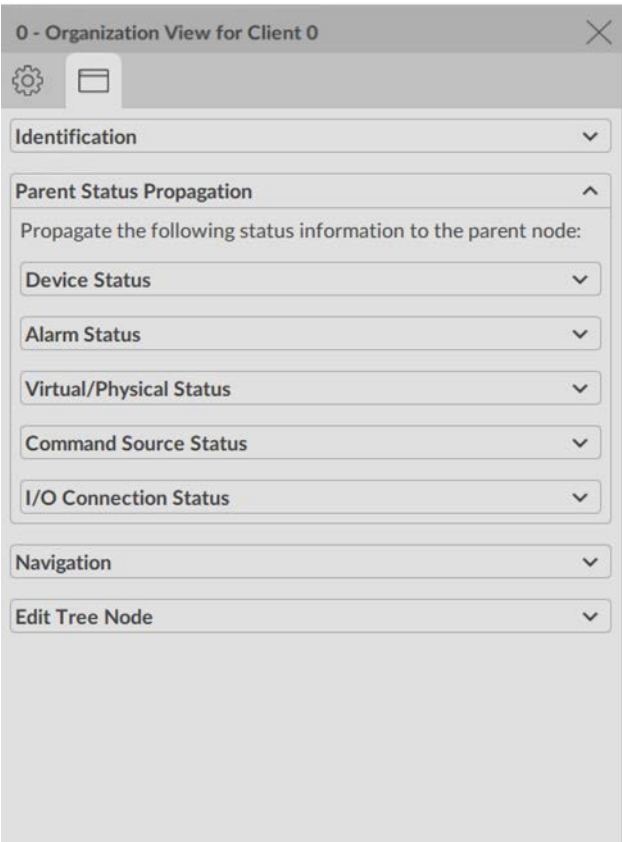
Command Source Commands

Line Level High Commands

Bypass/Check I/O Connection Commands

Group Commands

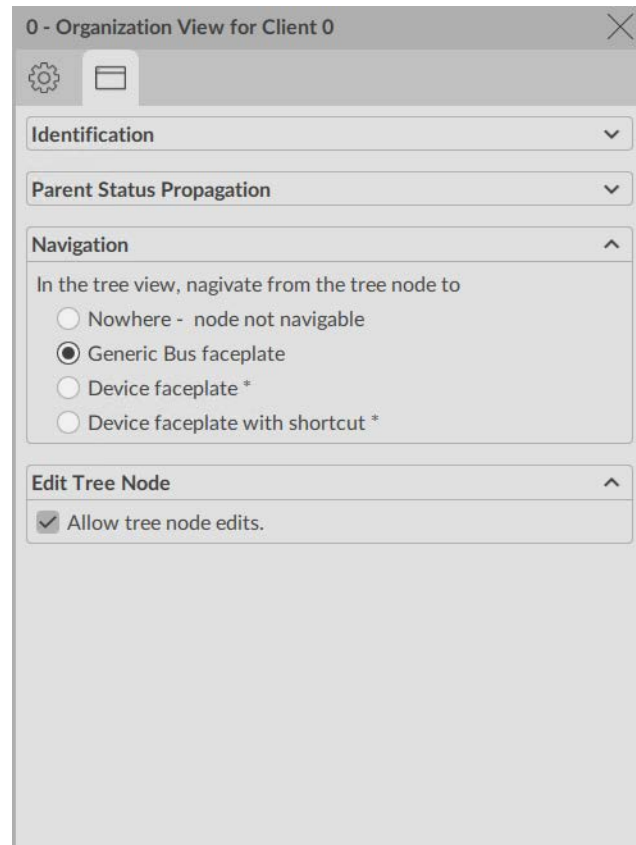
10. Define how to propagate status (Advanced HMI Tab - Parent Status Propagation).



11. Define navigation (Advanced HMI Tab - Navigation).

IMPORTANT

Navigation selections that are marked with an * use the Extended Tag Property @EngineeringUnit for the Bus Array instance (for example, Bus[37].@EngineeringUnit) because the @Navigation Extended Tag Property is not available in the user-defined data type (UDT) bus array.



Option	Description	Engineering Unit Extended Tag Property
Nowhere - node not navigable	There is no configured navigation for the node.	not used
Bus Faceplate	For use when the node does not have an associated display or faceplate. The generic faceplate can be configured to show any of the standard bus commands or status information.	not used
Device Faceplate	Show a standard library faceplate (or other faceplate that uses the same navigation method)	Device Tag. For example, "[PL5_20]MyPVLV"
Device Faceplate with Shortcut	Same with Device Faceplate	Device Tag. For example, "[PL5_20]MyraP_Opr_Unit"

Complete Organization Tree for this Example



The organization tree updates the Node array in the controller.

Node Element	Organization Tree Element
Node[0]	<root>
Node[1]	My Area
Node[2]	My Unit
Node[3]	T2 Unit

Set Start Node

You build the complete node tree once. Once you have a complete node tree, you can select which node to start for each client. For example, you can have one client view only the uTK100 portion and another client view only the uTK200 portion.

- 1. Select the start node.
- 2. Click the pencil button and click the Set Start Node button to define the start node.

For example, this view shows unit uTK100 as the start node. The other nodes are not viewable by this client.



Node Array Guidelines

Node array elements maintain the location of a single bus object in the organizational tree. Each entry that you make in the Node Tree by adding a child underneath a parent that child becomes an element in the Node array. Bus objects can exist as parents or children in multiple locations in the Node array.

- The maximum size of the node array is 3000 elements.
- Size the node array double the size of the bus array. This provides flexibility for expansion as well as assigning bus objects to more than a single node tree location.
- Do not change the node array manually within the Logix Designer application.
- Do not directly reference node array elements in logic. The node array location of a bus object can change. As children are added to the node tree, the nodes below those children will be shifted downward. As children are deleted, the nodes below will be

shifted upward. The raP_Opr_OrgDeviceCtrl object is provided in versions 5.30.00 of the library and later to allow a designer to manipulate nodes in the tree if necessary.

If you create your organization via the PlantPAx Configuration Tool, the Node array is correctly configured.

Node Tooltip Information

Tooltip information is provided when you hover over a node. When a bus object is owned, the tooltip will provide the name of the parent bus object that currently has ownership. When the Tree View edit toggle is "on" the tooltip text provides Node index, Bus Index, Current Owner Name, and Owner Bus Index information. When the Tree View edit toggle is "off" the tooltip text provides the Current Owner Name and the Bound/Unbound status information.

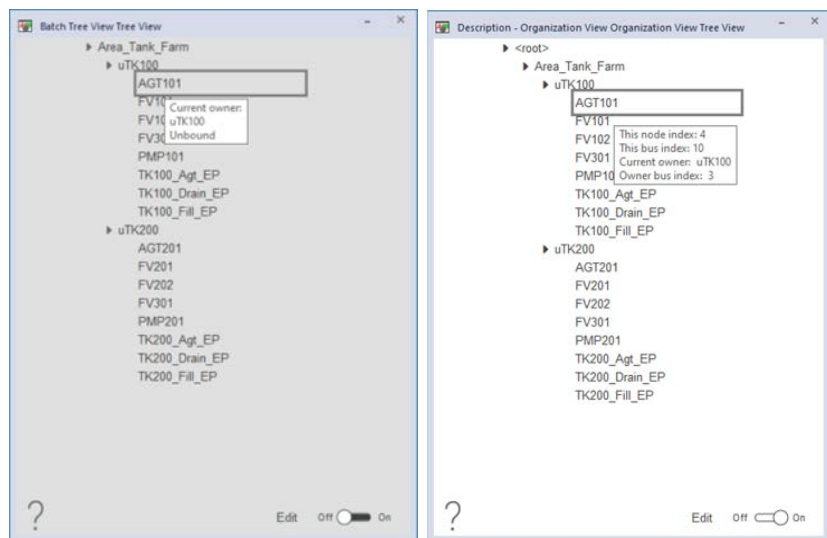
Additional context is provided in the following table:

Current Owner Tooltip Text	Description
None	The bus object is program ownable. It is not currently owned by a parent.
n/a	The bus object either does not have an associated command source or it has a command source with a program class that does not exist.
n/i	Ownership is suppressed at this node

A node status text is provided below the current owner to indicate when a node has been manipulated using an unbind or node command from the raP_Opr_OrgDeviceCtrl object.

Node Status Tooltip Text	Description
Unbound ⁽¹⁾	The bus object program ownership is unbound. This allows a user to place the object command source into operator mode as needed while still maintains ownership status.
DoNotOwn	Ownership at this node is suppressed.
Excluded	The object is unbound and excluded from organization accumulation at the parent.
DoNotPrp	Propagation of status and commands are suppressed at this node.

(1) Version 5.20.00 of the library provides tooltip indication for Bound/Unbound Status only. Releases before 5.20.00 do not provide node status tooltip text.














Status Indicators

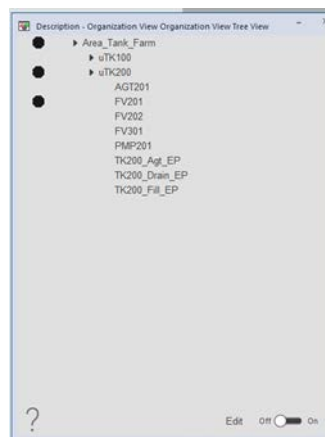
When configured to be available, specific status' can be viewed from any node in an organizational tree via the Bus faceplate. Select status' are represented by breadcrumbs, which appear in the organizational tree next to the nodes that are affected. Status' related to

alarms, command source, and virtualization are supported. The status' available on the faceplate are determined by the configuration previously entered.

When a condition occurs which produces a status point, those statuses may be relative to this object or any of its children. The active maximum priority alarm status symbol is displayed.

Status Symbol	Description
	This object or one of its children is not ready.
	Object 'not usable' by parent. Either the object is owned by another parent, or the object's command source helps prevent program control.
	This object or one of its children is alerting the operator (Attention.)
	This object or one of its children is in Virtual.
	This object or one of its children has an active Maintenance Bypass.
	This object or one of its children has an alarm inhibited. Only displayed if an active alarm is not present and no acknowledgment is required.
	This object or one of its children requires an alarm acknowledgment. Only displayed if an active alarm is not present.
	This object or one of its children has an active alarm with urgent priority.
	This object or one of its children has an active alarm with high priority.
	This object or one of its children has an active alarm with medium priority.
	This object or one of its children has an active alarm with low priority.

Example of the Not Ready symbol propagating up the tree from FV201 to the Area_Tank_Farm node:



Arbitration

The optional Arbitration function manages optional queues for shared devices when using the Owner function. It takes an ownership acquisition request, which is pending, and places it on the appropriate class queue. Any pending ownership request is presented to the ownership

function as a request in the order in which it resides on the queue. Once the current Owner releases its request, the next in the queue can take ownership.

The Arbitration queue:

- Enables user-defined arbitration rules to be applied to shared resources within a class.
- Handles and maintains multiple simultaneous requests for ownership.
- Manages a FIFO of the IDs of Ownership requests within a single command source class. (Operator, Program, External, Maintenance)

In the following example:

- TK100 and TK200 both have FV301 in their organization tree.



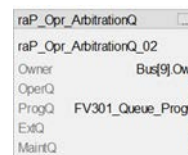
- Program an object for FV301 (Bus[9])



IMPORTANT For ownership commands and statuses to propagate, you must map the parameter pins as indicated in the example.

The parameter pins that are connected must align with the bus object index.

- There is an optional queue for each class request (Operator/Program/External/Maintenance). Currently, only Program is used, the rest are reserved for future functionality.



You can program logic to manipulate or reorder the entries on the individual queues. This queue shows ownership requests from Bus[3] (TK200) and Bus[2] (TK100).

• FV301_Queue_ProgQ	Local	[...]
▸ FV301_Queue_ProgQ[0]		3
▸ FV301_Queue_ProgQ[1]		2
▸ FV301_Queue_ProgQ[2]		0
▸ FV301_Queue_ProgQ[3]		0
▸ FV301_Queue_ProgQ[4]		0

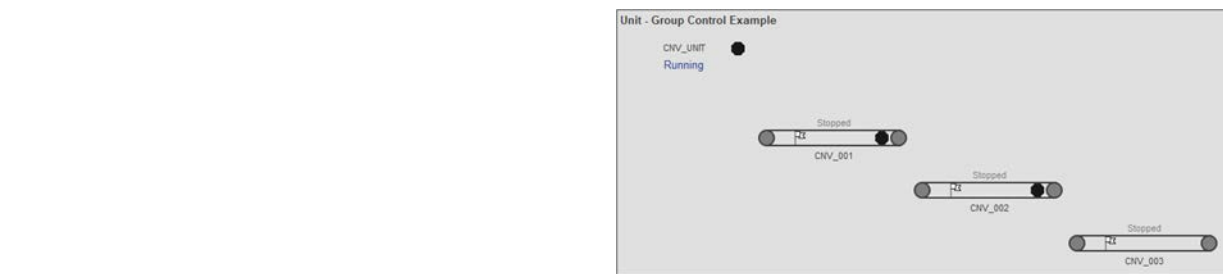
IMPORTANT	The user should not directly add or delete items on the individual queues. These functions are handled by the ownership functionality.
------------------	---

For object and visualization parameters, see Object and Visualization Parameters, [PROCES-RD201](#).

Unit Group Control

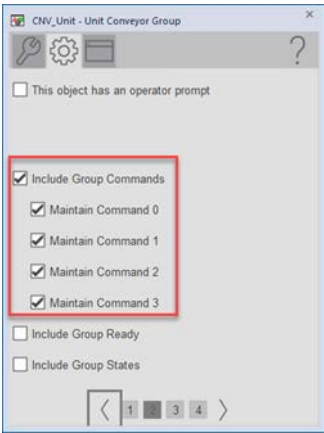
Use of the organization bus is a pre-requisite to using the Unit object. This is because the Unit object is designed to group children together and provide a propagation mechanism for aggregating status from child objects and issuing commands to child objects. Units can represent any group of equipment. This can be a tank within the S88 model, or it can represent a group of conveyors for a material handling application. The Unit provides four user-defined state commands that allow the designer to start and stop equipment as a group. The LLH commands for Emergency Stop, Software Stop, Permissive OK, and Interlock OK can be mapped to the children of the unit to prevent the grouped equipment from starting when the Unit object is not ready.

The following example demonstrates how a group of three conveyors can be controlled from the Unit faceplate. Depending upon the designer's preference, the equipment in the group can be started in a cascaded fashion or simultaneously. The unit in this example provides a conveyor start group command to start CNV_003 first and cascade back to CNV_001 as the downstream conveyors receive running feedback. A conveyor stop group command is provided to stop the conveyors in a similar fashion and a conveyor immediate stop group command is provided to stop all conveyors simultaneously.



The organizational structure is provided below. The conveyor equipment must be the immediate children of the Unit object. The node configuration for each object must be set to allow the propagation of the Line Level High Commands and the Group Commands for this functionality to work.

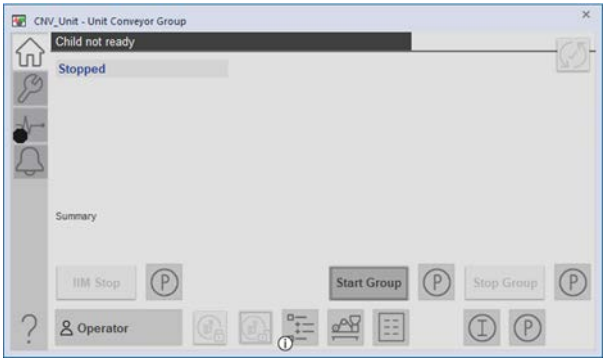
To enable the group command buttons on the Unit faceplate, configure the unit to include group commands from the engineering faceplate as well as have defined a description for each group command bit. The 'maintain command' configurations latch each group command request until the corresponding state status bit = 0. The 'include group ready' and 'include group states' configurations determine if the Unit objects will check the Inp_Sts or Inp_RdyOk inputs. When these settings are off, the Unit uses the bus inputs.



Name	Value	Force Mask	Style	Data Type	Description
• CNV_UnitXCmd	28000_000		Binary	SN7	Unit Conveyor Group External command to State (Bitmask)
CNV_UnitXCmd0	0		Decimal	BOOL	Stop Group
CNV_UnitXCmd1	0		Decimal	BOOL	Start Group
CNV_UnitXCmd2	0		Decimal	BOOL	Start Group
CNV_UnitXCmd3	0		Decimal	BOOL	IMM Stop

The configuration results in the following unit faceplate.

- Unit Command 1 (Bus cmd.28) – Stop Group
- Unit Command 2 (Bus cmd.29) – Start Group
- Unit Command 3 (Bus cmd.30) – Unused
- Unit Command 4 (Bus cmd.31) – Immediate Stop Group

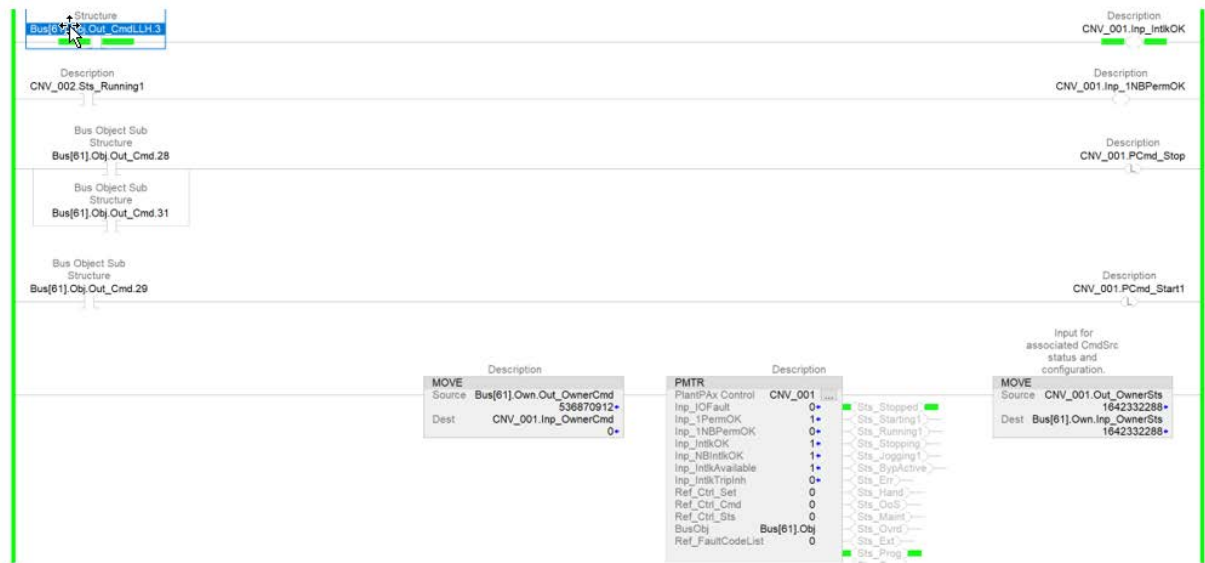


Unit Group Object Logic Example

The Unit conveyor group example uses the bus object assignments in the table below. The conveyor bus object elements must be referenced directly in the example logic due to the process object blocks not mapping the Unit line level high and state request bus commands and statuses internally. The group control bus commands and statuses are intended to provide the designer flexibility to define each state according to a wide variety of application requirements.

Bus Element	Name	Description
Bus[50]	CNV_Unit	Conveyor Group
Bus[61]	CNV_001	Conveyor 1
Bus[62]	CNV_002	Conveyor 2
Bus[63]	CNV_003	Conveyor 3

The following logic defines the motor input group logic for CNV_001. The Conveyor 2 Running status is mapped to the permissive OK input of conveyor 1 to ensure that the downstream conveyor starts first. Conveyor 3 is the most downstream conveyor and instead maps the Unit permissive OK status Bus[x].Obj.Out_CmdLLH.2.

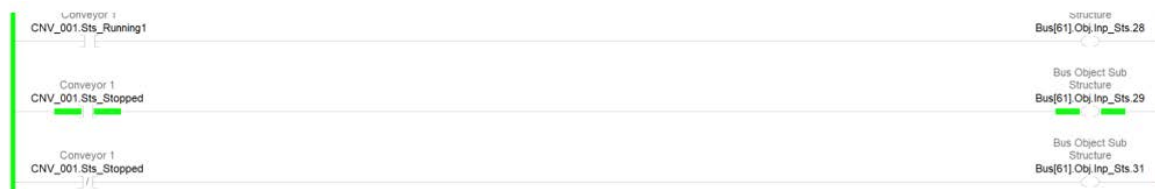


Similarly, the CNV_003 object checks for the CNV_001 and CNV_002 Sts.Stopped signal when a group stop is requested to allow for the most upstream conveyors to stop first. An immediate stop request would bypass this check.



Bus Output	Motor Input	Description
Bus[x].Obj.Out_CmdLLH.3	MTR.Inp_IntlkOK	Group Interlock OK
Bus[x].Obj.Out_Cmd.28	MTR.PCmd_Stop	Group Stop Command
Bus[x].Obj.Out_Cmd.31	MTR.PCmd_Stop	Group Immediate Stop Command
Bus[x].Obj.Out_Cmd.29	MTR.PCmd_Start1	Group Start Command
Bus[x].Obj.Out_CmdLLH.2	MTR.Inp_1NBPermOK	Group Permissive OK

The following logic defines the motor output group logic for CNV_001. It is important to note that the status mapping for each of these states is the inverse of the target state for the corresponding unit group command. The aggregated status at the Unit object is defined as "At least one child is not Started" so the conveyor stopped status is mapped to the bus input status. This is what controls the availability of the group commands from the Unit faceplate.



Motor Output	Bus Input	Description
MTR.Sts_Running1	Bus[x].Obj.Inp.Sts.28	Equipment not in State 1
MTR.Sts_Stopped	Bus[x].Obj.Inp.Sts.29	Equipment not in State 2
NOT MTR.Sts_Stopped	Bus[x].Obj.Inp.Sts.31	Equipment not in State 4

Hardware Organization Bus

The logic that is demonstrated in the previous organization example can be duplicated to create an organizational tree for the project hardware. The hardware organization bus functionality utilizes a set of hardware-centric Add-On Instructions that are bus capable. These Add-On Instructions are listed in the table below.

Hardware Bus Add-On Instruction	Description
raP_Dvc_LgxCPU_5x80 ⁽¹⁾	Logix CPU Utilization
raP_Dvc_Lgx_ChangeDet	Logix Change Detection
raP_Dvc_LgxRedun	Logix Redundant Controller Monitor
raP_Dvc_LgxTaskMon	Logix Task Monitor
raP_Dvc_LgxModuleSts ⁽²⁾	Logix I/O Module Connection Status

(1) raP_Dvc_LgxCPU_5x80 will not accept any bus commands from a parent or provide bus statuses as a child. It is intended to be used as the root object in the hardware organization tree. You can issue the commands that are listed below from the Bus Faceplate for the Logix CPU bus object to command all children accordingly.

(2) raP_Dvc_LgxModuleSts is the only hardware bus Add-On Instruction that uses the Not Ready (bit 8), Physical/Virtual (bits 10 and 11), and Bypassed/Check Connection (bits 26 and 27) status/commands described in the tables below. When used with the hardware bus organization, the Sts_IOfault output of this Add-On Instruction includes child module IO fault status.

The Hardware Bus provides:

- One-click alarm management commands for all I/O Modules that are connected to a single controller.
- One-click virtual mode or bypass mode commands for all I/O Modules that are connected to a single controller.
- Nearest point of failure error detection for connection paths.

The following statuses are propagated up the tree from the children to parent. These statuses are visible from the Hardware Tree View or the Bus Faceplate. Statuses 8, 9, 10, 11, 26, and 27 are specific to the raP_Dvc_LgxModuleSts.

HwBus[x].Out.Sts Bit	Status Produced and Propagated	Description
0	Alarms Active	At least one alarm is active for this object or its children
1	Alarms/Object to be Reset	At least one alarm is ready for reset for this object or its children
2	Ready for Reset	At least one object or child is ready for reset
3	Alarms Enabled	At least one alarm is enabled for this object or its children
4	Alarms Disabled	At least one alarm is disabled for this object or its children
5	Alarms Unsuppressed	At least one alarm is not suppressed for this object or its children
6	Alarms Suppressed	At least one alarm is suppressed for this object or its children
7	Alarms Shelved	At least one alarm is shelved for this object or its children

HWBus[x].Out_Sts Bit	Status Produced and Propagated	Description
8	Not Ready	I/O Module Connection Status is not "running" (Connected) and not bypassed
9	Bypass	Maintenance Bypass device status
10	Physical	At least one object or child is in Physical
11	Virtual	At least one object or child is in Virtual
26	Check Connection	This object or its children has a connection that is not bypassed
27	Bypassed	This object or its children has a connection that is bypassed

The IO Fault and Alarm Gate bus inputs are used within the raP_Dvc_LgxModuleSts Add-On Instruction. These inputs get propagated up the tree and allow the Add-On Instruction to set the Sts_IOFault when a child module has an IO Fault. The alarm gate is used to notify child modules that a parent node has already raised an alarm and it does not need to.

HWBus[x].Out_CmdLLH Bit	Status Issued and Propagated	Description
8	IO Fault	IO fault active
9	Alarm Gate	Gate child alarm when parent alarm is already active

The commands that are listed below can be issued from parent to child using the Generic Bus Faceplate. Commands 10,11,26, and 27 are specific to the raP_Dvc_LgxModuleSts.

HWBus[x].Inp_Cmd Bit	Commands Issued and Propagated	Description
3	Disable alarms	Disable all alarms
4	Enable alarms	Enable all alarms
5	Suppress alarms	Suppress all alarms
6	Unsuppress alarms	Unsuppress all suppressed alarms
7	Unshelve alarms	Unshelve all shelved alarms
10	Request Virtual	Request all to be in Virtual
11	Request Physical	Request all to be in Physical
26	IO Connection Bypass	Do not check IO Connection Status
27	IO Connection Check	Check IO Connection Status

Create the Hardware Organization Logic

Before you begin, import the process library Add-On Instructions into your controller project. All logic must be in one controller. The following Add-On Instructions are required:

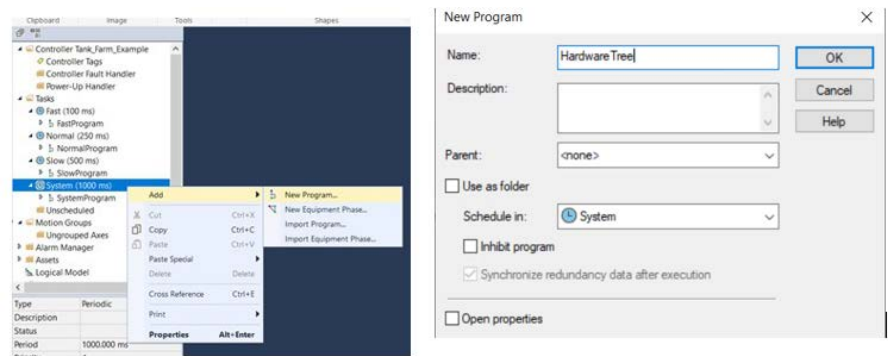
- raP_Opr_OrgScan Organizational Scan
- raP_Opr_OrgView Organizational View

IMPORTANT Application Code Manager and the PlantPax Configuration Tool provide simplified workflows that are more efficient to initially configure the hardware organization.

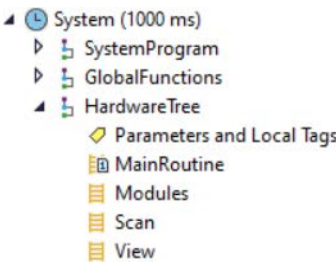
Create the HardwareTree Program

In a process controller project, create the HardwareTree program

1. Add a new program in the appropriate Task and name it HardwareTree. The default task is the System task.



2. Add a new routine to the HardwareTree program and name it MainRoutine. Repeat the process for a modules, scan, and view routine. Add JSR instructions to the MainRoutine for the modules, scan, and view routines.



3. Open the Scan routine and in the Ladder Diagram editor, add an raP_Opr_OrgScan Add-On Instruction to manage the propagation of commands/status and Ownership within the Hardware Organization Tree.

raP_Opr_OrgScan	
raP_Opr_OrgScan	?
BusNode	?
Bus	?

Create Array Tags

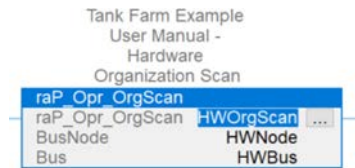
In the next steps, you create two array tags for these parameters that are used in this Add-On Instruction:

Bus array with an element for each piece of hardware in your I/O configuration. Make the array larger than the number of devices so you have room for future additions.

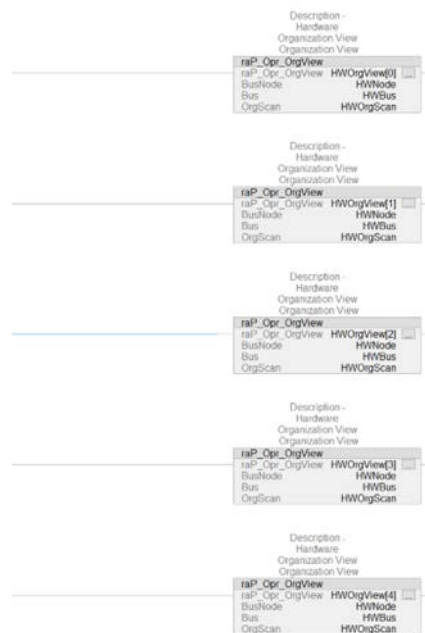
BusNode array with an element for each piece of hardware in the I/O configuration and its relationship to other hardware. Make the node array twice as large as the Bus array so that you have room for future additions

1. Right-click the raP_Opr_OrgScan ? and create a controller-scoped tag HWOrgScan of data type raP_Opr_OrgScan.

2. Right-click the BusNode ? and create a controller-scoped array tag HWNode of type raP_UDT_Opr_Bus_Node. The array tag must be named HWNode. Edit the array elements to have 100.
3. Right-click the Bus ? and create a controller-scoped array tag HWBus of type raP_UDT_Opr_Bus. The array tag must be named HWBus. Edit the array elements to have 50. Create more elements than your original list of bus elements to leave space to add future elements.



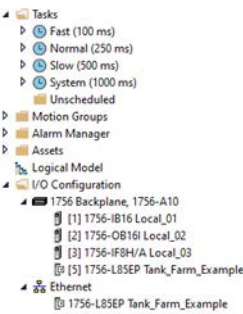
4. In the View routine and in the Ladder Diagram editor add an raP_Opr_OrgView Add-On Instruction for each HMI client that will be using the hardware organizational tree. This Example assumes five HMI clients so there are five individual rungs.
 - Right-Click the raP_OprView ? and create a tag HWOrgView of datatype raP_Opr_Orgview. Edit the Data Type and enter the array dimensions so that you have an element for each HMI Client.
 - Enter the BusNode, Bus, and Orgscan tags you created for the HWOrgScan instruction. Select one HWOrgView[x] element for each rung/instance to correspond with each HMI Client.



Define the HWBus Elements

Similar to the software bus example outlined in this document, each entity that is grouped into a hardware organization must be assigned a unique identifier. The HWBus array provides this mechanism, in addition to providing an interface to exchange commands and status.

Example I/O Configuration and Task Model

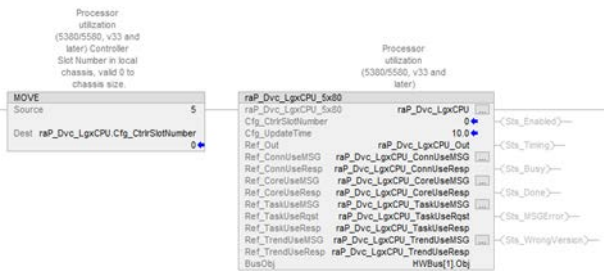


The elements can be in any order, and you can use any names that are appropriate for your application. The Change Detection and Redundancy Monitor Add-On Instructions were not used in this example.

Bus Element	Name	Description
Bus[0]		Reserved
Bus[1]	Tank_Farm_CLX	Controller
Bus[2]	Local_01	Local Controller Rack - Slot 1
Bus[3]	Local_02	Local Controller Rack - Slot 2
Bus[4]	Local_03	Local Controller Rack - Slot 3
Bus[5]	Task_Monitor_100ms	Fast Task Monitor
Bus[6]	Task_Monitor_250ms	Normal Task Monitor
Bus[7]	Task_Monitor_500ms	Slow Task Monitor
Bus[8]	Task_Monitor_1000ms	System Task Monitor

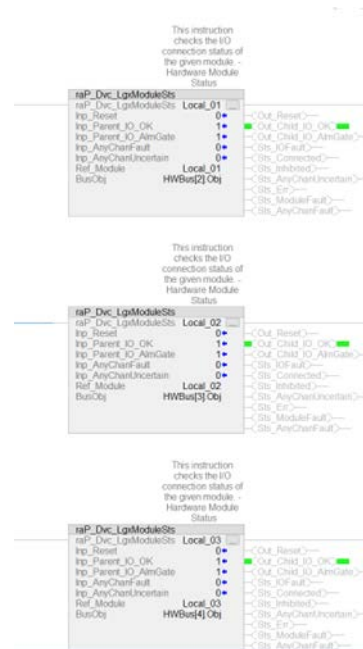
Configure the LCPU Instance

Import the PlantPax raP_Dvc_LgxCPU_5x80_5_30_00_Rung.L5X. This defines the Logix CPU Monitor. Assign HWBus[1].Obj to the BusObj InOut Parameter.



Configure the Module Status Instances

Import the PlantPAX raP_Dvc_LgxModuleSts_5_30_00_AOI.L5X. This creates the Logix module status Add-On Instruction. Create an instance of this Add-On Instruction for each IO Module in the I/O Configuration. Assign HWBus[2].Obj, HWBus[3].Obj, HWBus[4].Obj to the BusObj InOut Parameter for the Local_01, Local_02, and Local_03 instances respectively.



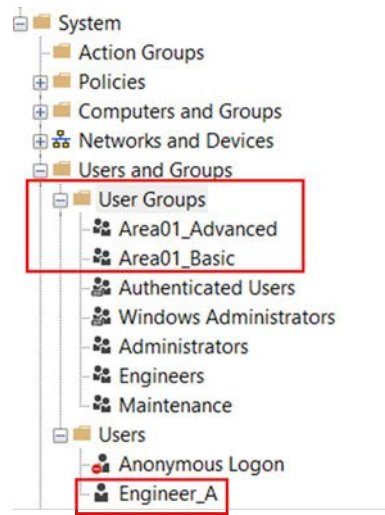
Configure the Task Monitor Instances

Import the PlantPAX raP_Dvc_LgxTaskMon_5.30.00.A01.L5X. This creates the Logix task monitor Add-On Instruction. Create an instance of this Add-On Instruction for each task in the project. Size the raP_Dvc_LgxTaskMon tag accordingly to the number of tasks in the project. Each task has an instance with raP_Dvc_LgxTaskMon[x]. Assign HWBus[5].Obj, HWBus[6].Obj, HWBus[7].Obj, HWBus[8].Obj to the BusObj InOut Parameter for the Fast, Normal, Slow, and System task instances respectively.



Create the Organizational Tree in the HMI Client

Make sure that you have the application configured with the appropriate user groups. This example uses the default Area01_Basic and Area01_Advanced user groups with a user Engineer_A.



Configure the Client Display

1. Select and copy the 'Hardware Tree' button from the (raP-5_30-SE) Graphic Symbols - Cross Functional.ggfx' global object file. Paste this button to the desired display.
2. Add the following lines to the client startup macro. The Template_ClientStartup macro file provides an example that can be uncommented and adjusted as needed. The template example uses the HWOrgView element 0 and /Area1/DATA:[Hardware] for the controller topic. Another startup macro must be created for each HMI client in the system so that each client can be assigned another HWOrgView element. The macros DefineShowTreeCmd and DefineShowHWTreeCmd are referenced in the startup macros and must exist in the project.

Line	Description
Define HW_RedefineShowTreeCmd DefineShowHWTreeCmd X	X = Client number (HWOrgView element) 0 in this example for client 0
HW_RedefineShowTreeCmd /Area/DATASERVER::[TOPIC]	Area = Area for data server DATASERVER = data server name TOPIC = shortcut name (path to controller) DATA:[Org.Example] in this example

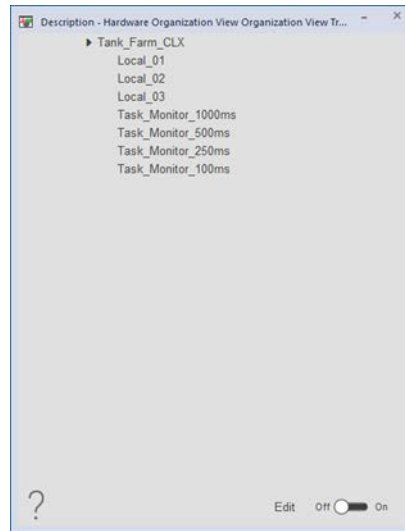
See [Client Startup Macro on page 81](#) to complete this configuration.

IMPORTANT The PlantPAx Graphic Framework Tool provides an interface for defining the client startup macros in the L1 configuration.

3. Generate the Client Display

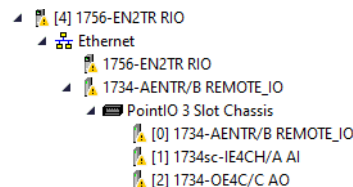
- Follow the steps detailed in the software organization example [Build the Node Tree \(FactoryTalk View\) on page 53](#) to add the HWBus objects to the hardware tree view. The Tank_Farm_CLX LCPU instruction is used as the parent object with the Logix module status and task monitor bus objects as children.

The hardware tree view organization appears similar to the following structure:

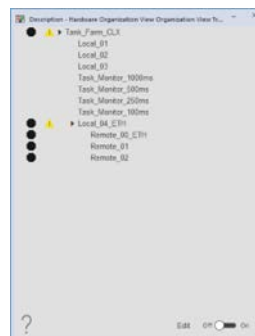


Hardware Tree Alarm Gating

Status and commands can be propagated through the tree for alarming and indications of individual module issues. Optional gating allows alarms from modules higher in the tree to gate alarms of modules lower in the tree to avoid 'nuisance' alarms. By propagating fault status downward through a tree hierarchy, a single fault status can be used to indicate any fault in the chain. This is useful for consolidating fault status when a module supplying input is located in a remote location. Consider the I/O configuration below. An Ethernet card is added to the I/O configuration from the previous hardware tree example for a remote POINT I/O™ rack.



The parent module in this case recognizes that the child modules are also in alarm and get those child alarm statuses in the hardware tree. The Local_04_ETH module status instance shows an in-alarm status for connection status and gate that status at the POINT I/O™ modules that are its children in the hardware tree.



Ownership (raP_Opr_Owner)

The raP_Opr_Owner (Ownership) Add-On Instruction extends the functionality of the PCMDSRC (Command Source) instruction to allow for ownership requests and owner ID book-keeping functionality.



For the object and visualization parameters, see PlantPAx Process Objects, publication [PROCES-RD200](#), and PlantPAx Visualization Files, publication [PROCES-RD201](#).

Guidelines

Use this instruction when it is desirable to maintain ownership IDs and manage ownership arbitration between the ownership classes (Opr, Prog, Ext, and Maint).

The raP_Opr_Owner functionality is included in the Bus Organizational UDT (raP_UDT_Opr_Bus). It is not necessary to create a separate raP_Opr_Owner instance to obtain ownership functionality between parent-child relationships that are configured in organizational trees that are processed by a raP_Opr_OrgScan instruction.

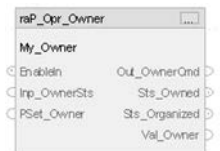
Functional Description

The raP_Opr_Owner Add-On Instruction is used to accept and process ownership requests by ID utilizing a PCMDSRC (Command Source) instruction for class arbitration rules. The basic class arbitration rules are implemented by the PCMDSRC instruction, which ownership requests are allowed, which ownership requests 'win' when multiple ownership requests are made by different classes of owners, and so on.

The raP_Opr_Owner instruction uses positive value DINTs as ownership IDs.

This instruction yields status as to the current owner IDs maintained if any. The ultimate 'winning' owner class and ID are also produced as status.

The state of 'Organization' is also indicated through status. This status indicates if the device/ object is in the correct PCMDSRC state for its ultimate owner and the status of any children if present and aggregated (that is, through the BUS organization). In this way you can determine if this device/ object is in the proper condition for operation.



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 Files

The raP_Opr_Owner_5.30.00_A01.L5X or (raP_Opr_Owner_5.20.00_A01.L5X) Add-On Instruction definition file must be imported into the controller project to be able to be used in the controller configuration. The maintenance release number (boldfaced) can change as maintenance revisions are created.

Visualization Files

The raP_Opr_Owner Instruction uses no visualization files or components.

Operations

Command Sources

The raP_Opr_Owner instruction has no commands or outputs that are intended to control equipment and therefore does not have any selection of active command source.

Alarms

The raP_Opr_Owner Instruction uses no alarms.

Virtualization

The raP_Opr_Owner Instruction has no Virtualization capability.

Execution

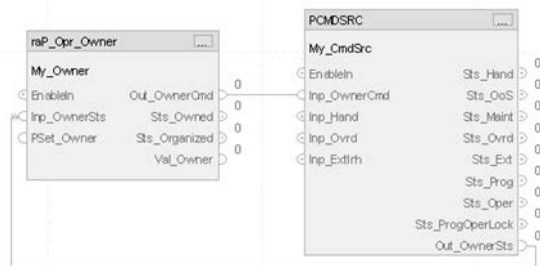
The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	The raP_Opr_Owner instruction clears all owner status and ID fields, and releases any ownership that is currently applied when scanned false or with the EnableIn=0.
Powerup (prescan, first scan)	The raP_Opr_Owner instruction clears all owner status and ID fields on PreScan/first scan.
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

The raP_Opr_Owner instruction must be coupled with a PCMDSRC instruction or a device/object that contains a PCMDSRC instruction. There are input and output parameters to accomplish the interface:



The 'Owner.Out_OwnerCmd' output parameter sends any pending ownership requests to the PCMDSRC owner interface parameter of the PCMDSRC (Inp_OwnerCmd).

The 'Owner.Inp_OwnerSts' input parameter receives existence, configuration, and current state information from the associated PCMDSRC (Out_OwnerSts).

When using a Process Object 5.00.xx and later device/object these parameters are supplied on the object to interface the ownership instruction with the object's internal PCMDSRC:




When using a Process Object 5.00.xx and later device/object as a participant on the Bus, the Bus referenced ownership interface parameters are used as input and output to the device/object:



Notes:

Arbitration (raP_Opr_ArbitrationQ)

The raP_Opr_ArbitrationQ (Arbitration) Add-On Instruction extends the functionality of the raP_Opr_Owner (Ownership) instruction to allow for the queuing of ownership requests within an ownership class.

 For the object and visualization parameters, see PlantPax Process Objects, publication [PROCES-RD200](#), and PlantPax Visualization Files, publication [PROCES-RD201](#).

Guidelines

Use this instruction if you want to extend the functionality of the raP_Opr_Owner to include multiple ownership requests within the same ownership class. One raP_Opr_ArbitrationQ instruction can be associated to a single raP_Opr_Owner to perform optional queuing of any of the four ownership classes (Oper, Prog, Ext, Maint).

Functional Description

The raP_Opr_ArbitrationQ Add-On Instruction is used to manage arrays of owner IDs for each class of ownership. Ownership requests made of the associated raP_Opr_Owner are intercepted by the raP_Opr_ArbitrationQ instruction and placed into a queue (DINT array) in the order in which they are received. By default, the earliest entry is used by the raP_Opr_Owner for ownership evaluation. As ownership requests and releases are made of the raP_Opr_Owner, the raP_Opr_ArbitrationQ instruction manages the addition and deletion of these requests and releases in the respective queues.

Use of the raP_Opr_ArbitrationQ instruction is optional. It extends the functionality of the raP_Opr_Owner instruction. Use the raP_Opr_ArbitrationQ instruction when there are multiple entities that could simultaneously request ownership of this entity AND you wish to maintain their order of request or manipulate the requests for prioritization.

Items in the queues can be reordered by user programming to accommodate prioritization schemes.

IMPORTANT

You should never add or delete IDs on the queue by user programming. Addition and deletion of IDs is done by the instruction itself based on the ownership requests made by the associated raP_Opr_Owner instruction.

The following image shows how a raP_Opr_ArbitrationQ instruction configured with the raP_Opr_Owner instruction 'My_Owner' as its associated owner instruction. Further, it is configured to have queues for Oper, Prog, and Maint owner classes. It does not use a queue for the External owner class:



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 Files

The raP_Opr_ArbitrationQ_5.30.00_AOI.L5X or (raP_Opr_ArbitrationQ_5.20.00_AOI.L5X) Add-On Instruction definition file must be imported into the controller project to be able to be used in the controller configuration. The maintenance release number (boldfaced) can change as maintenance revisions are created.

Visualization Files

The raP_Opr_ArbitrationQ Instruction uses no visualization files or components.

Operations

Command Sources

The raP_Opr_ArbitrationQ instruction has no commands or outputs that are intended to control equipment and therefore does not have any selection of active command source.

Alarms

The raP_Opr_ArbitrationQ Instruction uses no alarms.

Virtualization

The raP_Opr_Owner Instruction has no Virtualization capability.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	The raP_Opr_Arbitration instruction clears all queues and counters when scanned false or with the EnableIn=0.
Powerup (prescan, first scan)	The raP_Opr_Arbitration instruction clears all queues and counters on PreScan/first scan.
Postscan	No SFC Postscan logic is provided.

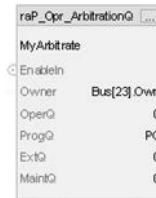
For more information, see the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#).

Tag Extended Properties and Default Alarm Settings

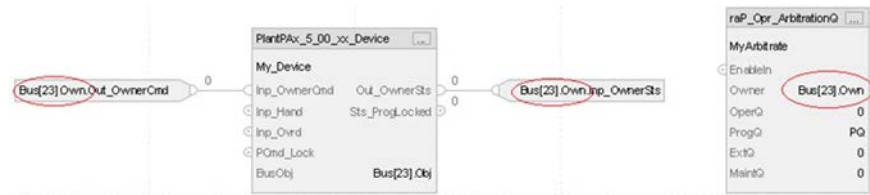
Common	
raP_Opr_ArbitrationQ.@Description	" "
raP_Opr_ArbitrationQ.@Area	"Area01"
raP_Opr_ArbitrationQ.@Instruction	" "
raP_Opr_ArbitrationQ.@Label	" "
raP_Opr_ArbitrationQ.@Library	"raP-5_30"
raP_Opr_ArbitrationQ.@URL	"n/a"

Programming Examples

The example in the Function Description section shows the basic use of the raP_Opr_ArbitrationQ Add-On Instruction for extending an ownership instruction. Typically, the raP_Opr_ArbitrationQ instruction is used in association with a Bus-resident entity. The following shows an arbitration instruction that is associated with a Bus referenced entity. The owner field is the 'Own' sub-element of the Bus structure:



A Bus enabled PlantPAx® device/object has its own Bus element. When extending the ownership functionality with the arbitration instruction, use the same Bus element reference that is used for that device/object:



Notes:

Organizational Scan (raP_Opr_OrgScan)

The raP_Opr_OrgScan (Organizational Scan) Add-On Instruction processes user-defined organizational trees to propagate status information from child nodes to parent nodes, and to propagate commands from parent nodes to child nodes. Further ownership requests and status can be propagated between parent and child nodes. The functionality to edit any organizational trees is built into this Add-On Instruction and edit requests are executed synchronously with the organizational scan.



For the object and visualization parameters, see PlantPax Process Objects, publication [PROCES-RD200](#), and PlantPax Visualization Files, publication [PROCES-RD201](#).

Guidelines

Use this instruction if you want to build parent-child relationships between controller-resident entities and propagate status, command, and ownership functionality between them.

Functional Description

The raP_Opr_OrgScan Add-On Instruction is used to propagate the information between elements of organizational trees and allow ownership relationships between those elements. It also maintains the organizational tree editing functions and the edit token ownership.

A single raP_Opr_OrgScan instruction is to be used to scan all organizational trees in a controller. As such, a single instance of the Add-On Instruction is to be scanned unconditionally in a slow, low- priority task.

Organization Scan		
raP_Opr_OrgScan		
raP_Opr_OrgScan	OrgScan	...
BusNode	Node	
Bus	Bus	

'Node' is an array that is composed of elements of type 'raP_UDT_Opr_Bus_Node'. This array must be of sufficient length to accommodate the maximum possible number of organizational tree nodes. Typical systems can have 100...1000 nodes depending upon the complexity of the organizational trees. Significant scans can occur when Node arrays with greater than 500 elements are used.

'Bus' is an array that is composed of elements of type 'raP_UDT_Opr_Bus'. This array must be of sufficient length to accommodate the maximum possible number of devices/objects that you wish to place on the Bus.

IMPORTANT	The name of the Node array must be 'Node' and the name of the Bus array must be 'Bus' for raP_Opr_OrgView operation.
------------------	--

Edit functionality and Edit Token management is also maintained in the raP_Opr_OrgScan instruction. All editing of the nodal organizational trees occurs through this 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 Files

The raP_Opr_OrgScan_5.30.**00**_A01.L5X or (raP_Opr_OrgScan_5.20.**00**_A01.L5X) Add-On Instruction definition file must be imported into the controller project to be able to be used in the controller configuration. The maintenance release number (boldfaced) can change as maintenance revisions are created.

Visualization Files

The raP_Opr_OrgScan Instruction uses no visualization files or components.

Operations

Command Sources

The raP_Opr_OrgScan instruction has no commands or outputs that are intended to control equipment and therefore does not have any selection of active command source. A raP_Opr_Owner and PCMDSRC instances are present in the Add-On Instruction to facilitate edit token ownership by an HMI client through a raP_Opr_OrgView instance.

Alarms

The raP_Opr_OrgScan Instruction uses no alarms.

Virtualization

The raP_Opr_OrgScan Instruction has no Virtualization capability.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	No EnableIn False logic is provided. The raP_Opr_OrgScan instruction must always be scanned true. In relay ladder logic, the raP_Opr_OrgScan instruction must be by itself on an unconditional rung.
Powerup (prescan, first scan)	All status and internal limits are cleared on prescan/first scan and array bounds and existing node configuration are checked.
Postscan	No SFC Postscan logic is provided.

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

Tag Extended Properties and Default Alarm Settings


Common	
raP_Opr_OrgScan.@Description	" "
raP_Opr_OrgScan.@Area	"Area01"
raP_Opr_OrgScan.@Instruction	" "
raP_Opr_OrgScan.@Label	" "
raP_Opr_OrgScan.@Library	"raP-5_30"
raP_Opr_OrgScan.@URL	"n/a"

Programming Examples

The example in the Function Description section shows the basic use of the raP_Opr_OrgScan Add-On Instruction. The raP_Opr_OrgScan can be executed from any controller language. But must be executed unconditionally. The scan update of all nodes in a system may require long scan times, therefore it is recommended to be executed from a slow, low-priority task. Further, the associated timeouts (Program, and so on) should be lengthened accordingly.

Organizational View (raP_Opr_OrgView)

The raP_Opr_OrgView (Organizational View) Add-On Instruction continuously scans the organizational trees and queues the information into a standard hierarchical tree view for presentation on a single HMI (FactoryTalk® View SE) client.

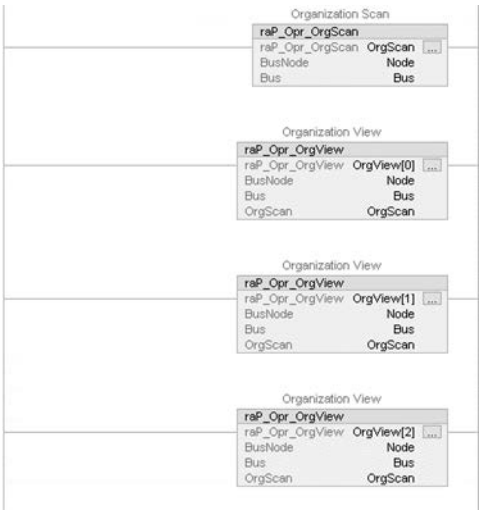
 For the object and visualization parameters, see PlantPax Process Objects, publication [PROCES-RD200](#), and PlantPax Visualization Files, publication [PROCES-RD201](#).

Guidelines

Use this instruction if you want to display organizational tree information on an HMI client. Another instance of the raP_Opr_OrgView instruction must be instantiated and scanned within the user project for each intended HMI client.

Functional Description

The raP_Opr_OrgView Add-On Instruction is used to read and format organizational tree information into a standard tree view on the HMI. The Add-On Instruction automatically adjusts the tree view based on user edits of the organizational tree. It is intended to have one raP_Opr_OrgView instruction instance for each HMI client viewing the organizational trees in a controller. This is so each HMI client can have a tree view that is unaffected by the actions of another client (expand, collapse, edit, and so on).



Here there are three raP_Opr_OrgView instances that are associated with the primary raP_Opr_OrgScan instance to update three individual HMI clients.

IMPORTANT An array of raP_Opr_OrgView backing tags must be created at the controller scope of name 'OrgView.' Each element of the array is used as the backing tag for each instance servicing a single HMI client. See Organizational Scan (raP_Opr_OrgScan) on page 143 for other naming requirements.

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 Files

The raP_Opr_View_5.30.**00**_A0I.L5X or (raP_Opr_View_5.20.**00**_A0I.L5X) Add-On Instruction definition file must be imported into the controller project to be able to be used in the controller configuration. The maintenance release number (boldfaced) can change as maintenance revisions are created.

Visualization Files

See [Library Versions on page 15](#) for general information on visualization files.

Operations

Command Sources

The raP_Opr_OrgView instruction has no commands or outputs that are intended to control equipment and therefore does not have any selection of active command source.

Alarms

The raP_Opr_OrgView Instruction uses no alarms.

Virtualization

The raP_Opr_OrgView Instruction has no Virtualization capability.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	No EnableIn False logic is provided. The raP_Opr_OrgView instruction must always be scanned true. In relay ladder logic, the raP_Opr_OrgView instruction must be by itself on an unconditional rung.
Powerup (prescan, first scan)	All status, internal HMI buffers, and internal limits are cleared on prescan/first scan and array bounds and existing node configuration are checked.
Postscan	No SFC Postscan logic is provided.

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

Tag Extended Properties and Default Alarm Settings

Common	
raP_Opr_OrgView.@Description	" "
raP_Opr_OrgView.@Area	"Area01"
raP_Opr_OrgView.@Instruction	" "
raP_Opr_OrgView.@Label	" "
raP_Opr_OrgView.@Library	"raP-5_30"
raP_Opr_OrgView.@URL	"n/a"

Organizational Node Interface (raP_Opr_OrgDeviceCtrl)

The raP_Opr_OrgDeviceCtrl Add-On Instruction extends the functionality of the raP_Opr_OrgScan (Organizational Scan) and raP_Opr_Owner (Ownership) instructions to provide program inputs to suppress ownership, suppress propagation, or exclude a child from organized status accumulation at the child level.



For the object and visualization parameters, see PlantPax Process Objects, publication [PROCES-RD200](#), and PlantPax Visualization Files, publication [PROCES-RD201](#).

Guidelines

Use this instruction to optionally extend OOAP functionality by manipulating the ownership or propagation behavior at a single child in the organizational tree.

This instruction utilizes the parent bus index and child bus index inputs to find the relational node in the tree where that parent-child relationship exists. When edits to the tree occur which result in the shifting of the node array data, this instruction detects those edits and update the relational node accordingly. Any active suppression or exclusion being asserted on the relational node will be maintained during tree edits and will shift as needed.

An instance of the raP_Opr_OrgDeviceCtrl instruction can be used for each parent/child relationship that is defined in the organizational tree to manage a given node and conditionally perform any combination of the following functions on the child:

- Do not own this child when ownership is requested by the parent.
- Suppress all command and status propagation to and from a child.
- Unbind ownership at the child and exclude it from children organized status aggregation at its parent.

The ownership and propagation functions can be useful in cases where a device or equipment group is shared among multiple parents but does not need to be owned in all situations. Suppressing ownership while still propagating status from the child allows that parent to monitor that child while still letting other parents claim ownership and use it as needed. If a child's node should be ignored completely, ownership and propagation can be suppressed simultaneously. With these functions, if the child object goes into an alarm or not ready state, the parent is not affected. This can often be the case with certain parent objects that can be configured to stop or hold on to a child in alarm or not useable. When a child's ownership is suppressed by a parent, it is no longer included in the accumulation of the Sts_ChildrenOwned and Sts_ChildrenOrganized outputs at the parent.

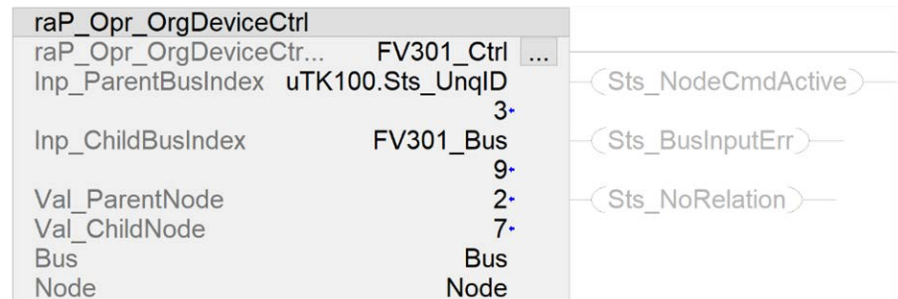
The exclusion function is valuable in scenarios where an operator may be required to put a child device into operator mode to adjust setpoints (PVSD, PPID, etc.). While excluded, the child will not be included in the aggregation of the parent's children organized status allowing the operator to place the object into operator mode and adjust setpoints. Excluding a child from organization at the node with the raP_Opr_OrgDeviceCtrl Add-On Instruction only excludes that child from the organized status of the parent to that node. A parent still retains ownership of the child while the child is excluded from organization.

If the Bus[ChildObjectId].Inp_ProgDoNotInclude is used to exclude the child from organization then that child is excluded from organization accumulation in every location it exists in the tree (all nodes). Refer to the unbinding ownership section of chapter 4 in this document for more detail.

Functional Description

Use of the raP_Opr_OrgDeviceCtrl instruction is optional. It extends the OOAP functionality that is defined in chapter 4. This Add-On Instruction is used to alter the ownership and propagation behavior of a single node in the organizational tree. Multiple instances of the raP_Opr_OrgDeviceCtrl Add-On Instruction can be defined to manage the behavior of multiple nodes in the organization as needed.

The following image shows how a raP_Opr_OrgDeviceCtrl instruction is configured with the parent and child bus index references. When the parent or child object is an raP_Opr_Area, raP_Opr_Unit, raP_Opr_EMGen, or raP_Opr_EPGen object the Sts_UnqID output parameter can be used as the bus index input. The designer can explicitly use the bus index number of the desired parent or child instead of a tag as well.



The following program command inputs are available for the controls designer to set in logic as needed to control the behavior of the selected node. All nodes propagate commands and status, accept ownership, and are included in organization by default.

Node Commands ⁽¹⁾	Description
Pcmd_SuppressOwn	1 = Suppress Ownership at Child
Pcmd_UnsuppressOwn	1 = Unsuppress Ownership at Child. Node is included in ownership by default unless a Pcmd_SuppressOwn command has been given
Pcmd_Exclude	1 = Unbind and Exclude Child from Organization Status of Parent
Pcmd_Include	1 = Bind and Include Node into Organization Accumulation. Node is included by default unless a Pcmd_Exclude command has been given
Pcmd_DoNotPropagate	1 = Suppress propagation of Status and Commands From/To Parent
Pcmd_Propagate	1 = Unsuppress Propagation of Status and Command form/to the child. Node propagates by default unless a Pcmd_DoNotPropagate command has been given

(1) For more information on unbinding ownership and excluding from organization, refer the [Unbinding Ownership](#) and [Exclusion](#) sections in Chapter 4.

The following status outputs are provided:

Status	Description
Sts_NodeCmdActive	1 = Node Ownership Suppression, Organization Exclusion, or Propagation Suppression Active
Sts_SuppressOwn	1 = Ownership at child node is being suppressed for the parent node
Sts_Exclude	1 = All inclusion of Ownership and Status For Parent/Child Bus Pair Suppressed at the Child Node
Sts_DoNotPropagate	1 = Propagation of Alarms and Status For Parent/Child Bus Pair Suppressed at the Child Node
Sts_BusInputErr	1 = Parent or Child Bus Index OOR
Sts_NoRelation	1 = No Relationship Found for Parent/Child Bus Pair in the Organization Tree

IMPORTANT

The raP_Opr_OrgDeviceCtrl Add-On Instruction supports only a single instance of a parent/child relationship in the node tree. Adding a parent with all its children in the organizational tree more than once will create multiple nodes that maintain the same parent-child relationship. Doing so results in unpredictable functionality of this Add-On Instruction. The raP_Opr_OrgDeviceCtrl Add-On Instruction is capable of only finding a single node.

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 Files

The raP_Opr_OrgDeviceCtrl_5.30.**00**_Add-On Instruction.L5X or (raP_Opr_OrgDeviceCtrl_5.20.**00**_A0I.L5X) Add-On Instruction definition file must be imported into the controller project to be able to be used in the controller configuration. The maintenance release number (boldfaced) can change as maintenance revisions are created.

Visualization Files

The raP_Opr_OrgDeviceCtrl Instruction uses no visualization files or components.

Operations

Command Sources

The raP_Opr_OrgDeviceCtrl instruction has no commands or outputs that are intended to control equipment and therefore does not have any selection of active command source.

Alarms

The raP_Opr_OrgDeviceCtrl Instruction uses no alarms.

Virtualization

The raP_Opr_OrgDeviceCtrl Instruction has no Virtualization capability.

Execution

The following table explains the handling of instruction execution conditions.

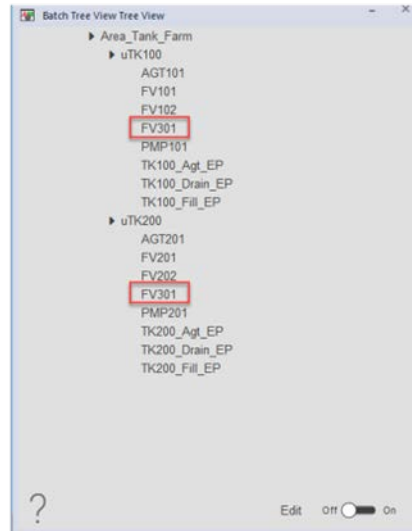
Condition	Description
EnableIn False (false rung)	The raP_Opr_OrgDeviceCtrl instruction continues to monitor the node array for changes and shift relational node to be suppressed as needed.
Powerup (prescan, first scan)	No Powerup prescan logic is provided.
Postscan	No SFC Postscan logic is provided.

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

Programming Examples

Shared Child Device Example

Consider the tree structure example from Chapter 4. The objects uTK100 (Bus 3) and uTK200 (Bus 4) share the object FV301 (Bus 9) as a child. Neither uTK100 nor uTK200 in this case require FV301 during the Agitate or Fill phases. The raP_Opr_ArbitrationQ instruction can be implemented to allow the opposite tank to wait in a FIFO queue to gain ownership of FV301 when the other tank has completed. If this is not desirable, the control designer can selectively own the children under each tank based on the action that is required instead.



The FV301_Ctrl instance of the raP_Opr_OrgDeviceCtrl Add-On Instruction below identifies node 7 that maintains the parent-child relationship between uTK100 and FV301. Selectively owning FV301 during the fill and agitate states allows TK200 to go into the Drain state without having to wait for TK100 to complete.

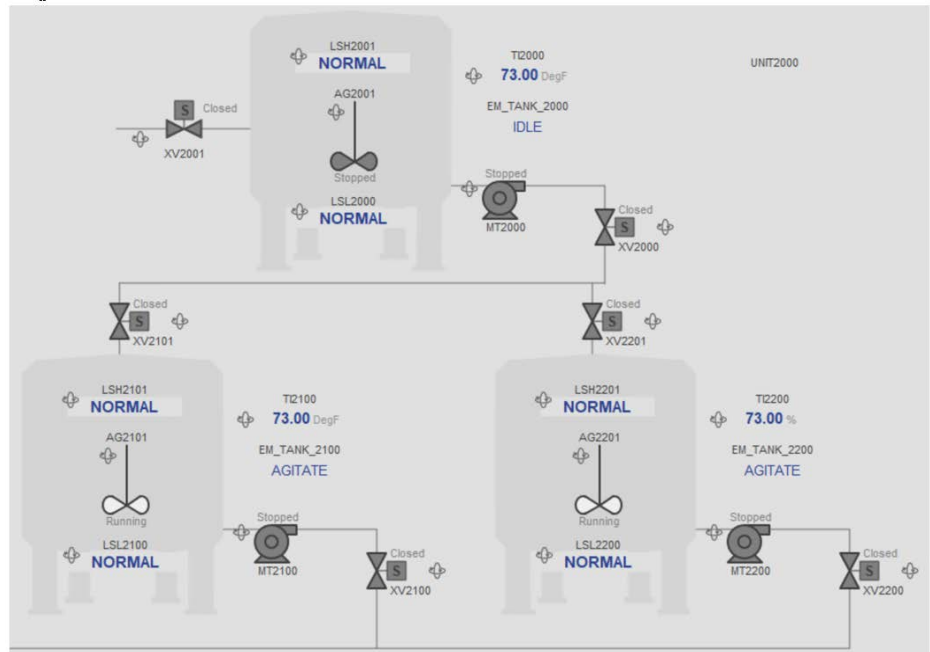
An identical program-scoped instance of FV301_Ctrl can also be defined in the logic for uTK200 to be used to selectively own FV301 based on its requested state.

If an engineer were to edit the tree in a running system and add a new node for a Tank 300 unit under the Area_Tank_Farm parent, then all child nodes underneath Tank 100 and Tank 200 shift downward in the array to accommodate that addition. The raP_Opr_OrgDeviceCtrl AOI will detect edits such as this and update the FV301_Ctrl.Val_ChildNode accordingly. Any active suppression will be updated and shifted as well when the FV301_Ctrl.Val_ChildNode is updated.

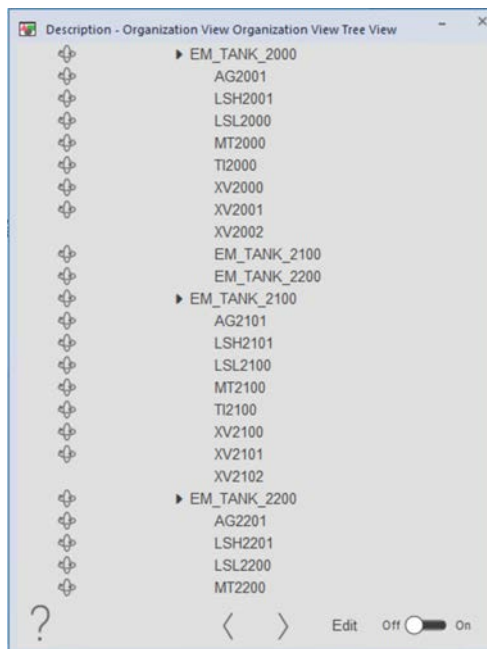


Selectively Owning a Parent

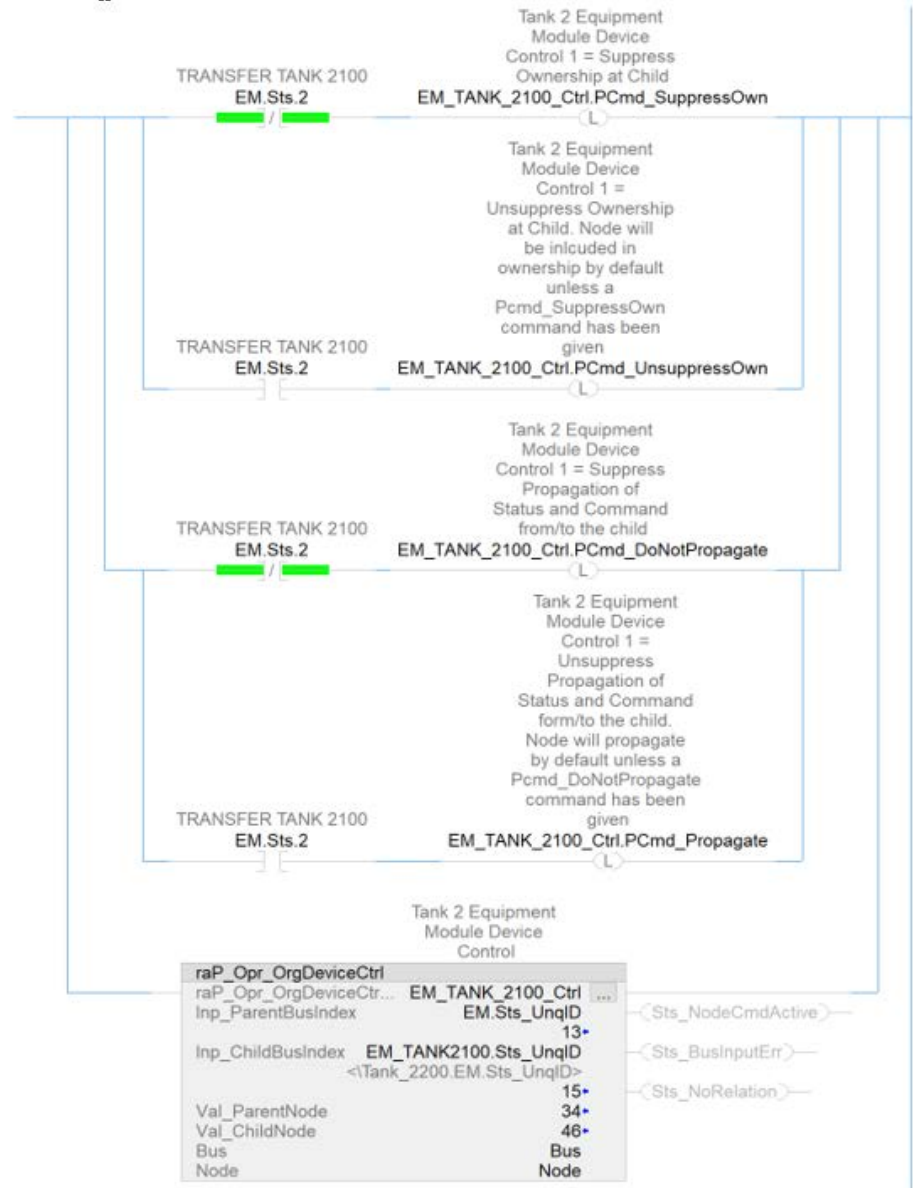
The second example below illustrates how the ownership suppression function can be used on another parent in the tree. Tank 2000 below (Pictured at the top) can transfer to either Tank 2100 or Tank 2200. If Tank 2000 is actively transferring to Tank 2100, then Tank 2200 must be left available to transfer out to the next area (Tank 3000).



By defining the tree hierarchy as shown below, the designer can give Tank 2000 the option to only own EM_TANK_2100 or EM_TANK_2200 depending on which transfer state it is in. Leaving EM_TANK_2200 unowned allows it to simultaneously transfer out to another tank. EM_TANK_2100 can be placed as a child underneath another parent without its children being redefined in the tree again underneath it.

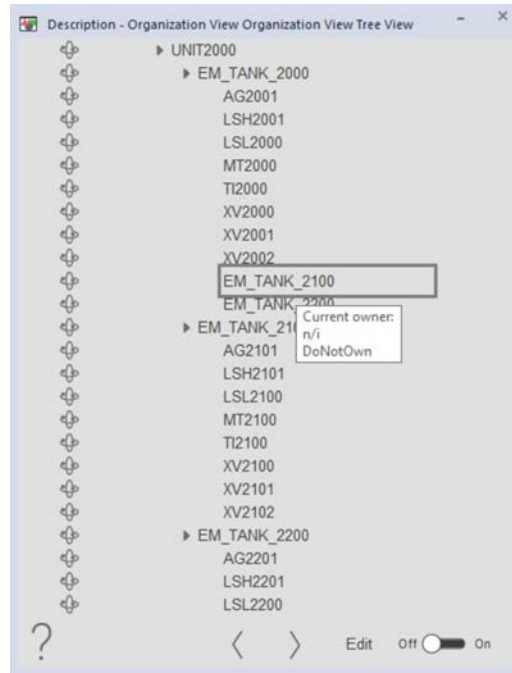


The logic below is defined in the EM_TANK_2000 program to suppress ownership and propagation of the Tank 2100 equipment module when the Tank 2000 equipment module is not in the transfer to Tank 2100 state.



When the EM_TANK_2100 child has ownership suppressed it is indicated with "DoNotOwn" on the Tree View tooltip display when hovering over that node. Similarly, when a node is excluded, the tooltip will indicate "Excluded" and if status and command propagation is suppressed the

tooltip will indicate “DoNotPrp”. The ownership-suppressed tooltip indication takes precedence over all other node status texts.



Limitations

The use of the raP_Opr_OrgDeviceCtrl is dependent on the design of the organizational tree. In both examples above, the Add-On Instruction is used to find a single node in the tree that can then be manipulated as needed. If that parent-child relationship is added to another location in the tree in either case, then there would be two nodes in the tree maintaining the same relationship. The raP_Opr_OrgDeviceCtrl would not be capable of suppressing that second node resulting in undesirable functionality. Consideration should be made during the design of the tree to prevent such cases. In the second example, the EM_TANK_2100 and EM_TANK_2200 bus objects are added as children underneath EM_TANK_2000 to more easily manage tank transfers. These bus objects were added as nodes without their children in this extra location due to this limitation. Not re-adding each child of EM_TANK_2100 again underneath the EM_TANK_2000 parent allows the designer the capability to still use the raP_Opr_OrgDeviceCtrl Add-On Instruction with EM_TANK_2100 devices (AG2101, XV2100, etc.). This approach also has the advantage of reducing the number of nodes that are used in the system.

Process Area Module (raP_Opr_Area)

The raP_Opr_Area object groups Units together, and provides a propagation mechanism for aggregating status from Unit objects, and broadcasting commands to Unit objects.



For the object and visualization parameters, see PlantPAx Process Objects, publication [PROCES-RD200](#), and PlantPAx Visualization Files, publication [PROCES-RD201](#).

Guidelines

The Area group is based in a controller.

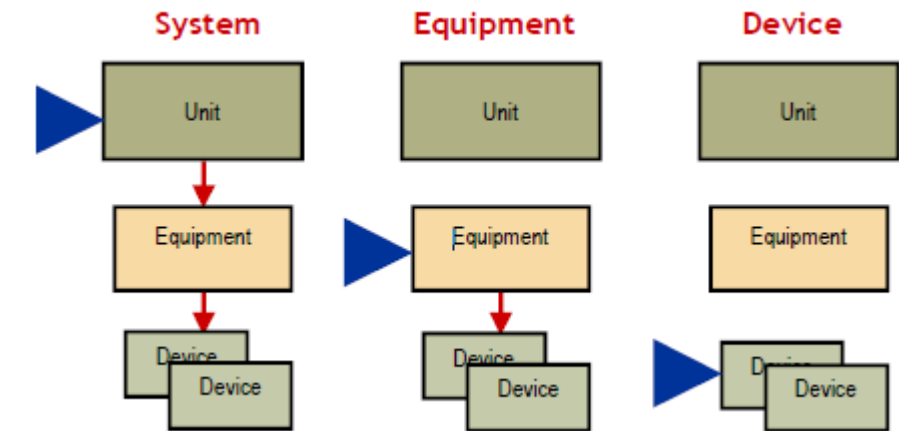
The Area is responsible for managing the equipment that is associated to that Area. These responsibilities include, but are not limited to, the following:

- Command Source management for a group of equipment.
- Alarm management for a group of equipment.
- Aggregate (propagation) status (for items such as: command source, alarm, configuration errors, and so forth) and provided “bread crumbs” for navigation.
- Provide broadcast (propagation) command mechanism.
- Detects failure conditions, such as Emergency Stop and Software Stop.
- Provides mechanism for extended Alarms.
- Monitor various Area failure conditions, and produce alarms.
- Provide a propagation mechanism to allow the Area to receive status from and send commands to a group of equipment.
- Provides the ability to produce a Software Stop condition based on any of the following:
 - Alarm from any lower-level object
 - Software Stop input
 - Area Alarm

Functional Description

Command Source Management

Allows you to interact with the system at various levels.



Use the PCMDSRC PlantPax® instruction to manage the command source (owner) of an instruction or control strategy. For more information, see PlantPax Process Control Instructions, publication [PROCES-RM215](#).

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 Files

The raP_Opr_Area_5.30.**00**_A0I.L5X or (raP_Opr_Area_5.30.**00**_A0I.L5X) Add-On Instruction must be imported into the controller project to be used in the controller configuration. The maintenance release number (boldfaced) can change as maintenance revisions are created.

Visualization Files

See [Library Versions on page 15](#) for general information on visualization files.

Operations

Command Sources

The raP_Opr_Area instruction has no commands or outputs that are intended to control equipment and therefore does not have any selection of active command source.

Alarms

The raP_Opr_Area Instruction uses the following alarms, which are implemented by using Tag Based Alarms.

Alarm	Alarm Name	Description
E-Stop trip	Alm_EStopTrip	Raised when an emergency stop condition triggers a change in state of the Area.
S-Stop trip	Alm_SStopTrip	Raised when a software stop condition triggers a change in state of the Area.

Virtualization

The raP_Opr_Area Instruction has no Virtualization capability.

Execution

The handling of instruction execution conditions.

Condition	Description
EnableIn False (False Rung)	Handle processing for EnableIn False (False Rung) the same as if the Area were Disabled by Command. The Area outputs are de-energized and the Area is shown as Disabled on the HMI.
Powerup (Pre-scan, First Scan)	Handles processing of command source and alarms on Pre-scan and Powerup. On Powerup, the Area is treated as if it were Commanded to reset all program and operator commands
Postscan (SFC Transition)	No SFC Postscan logic is provided.

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

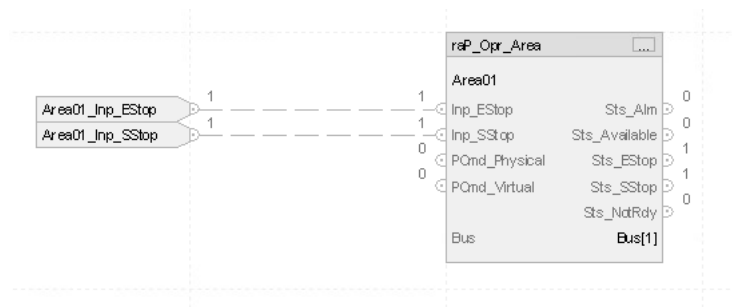
Tag Extended Properties and Default Alarm Settings

Common	
raP_Opr_Area.@Description	"Area"
raP_Opr_Area.@Area	"Area01"
raP_Opr_Area.@Instruction	"raP_Opr_Area"
raP_Opr_Area.@Label	"Area"
raP_Opr_Area.@Library	"raP-5_30"
raP_Opr_Area.@URL	"n/a"
raP_Opr_Area.Cfg_HasMoreObj.@Navigation	" "
General	
raP_Opr_Area.Sts_ExtddAlms.@Label	"Extended alarm"

Alarms		Alarm Default Message	Severity
raP_Opr_Area.Sts_EStopTrip.@Label	"Emergency stop"	"/*S:0 %.@Description*/: Emergency stop	750
raP_Opr_Area.Sts_SStopTrip.@Label	"Software stop"	"/*S:0 %.@Description*/: Software stop	750

Programming Example

The example in the Functional Description section shows the basic use of the raP_Opr_Area Add-On Instruction. Typically, the raP_Opr_Area instruction is used in association with a Bus-resident entity. The following example shows an area instruction that is associated with a Bus referenced entity. The area is also connected to two inputs, emergency stop, and software stop that can trigger two alarms. The area object is typically used in an S88 application but can be applied to suit numerous hierarchy layouts.



Notes:

Process Unit (raP_Opr_Unit)

The raP_Opr_Unit object groups Equipment together, and provides a propagation mechanism for aggregating status from Equipment, and broadcasting commands to Equipment. As an example each vessel, tank, mixer, machine, etc... within the control system would be considered a Unit.

- Units are presumed to operate on only one batch at a time.
- Units operate relatively independently of one another.
- This term applies to both the physical equipment and the equipment entity.
- Examples of major processing activities are; react, crystallize, and make a solution.



For the object and visualization parameters, see PlantPAx Process Objects, publication [PROCES-RD200](#), and PlantPAx Visualization Files, publication [PROCES-RD201](#).

Guidelines

The raP_Opr_Unit (Process Unit) object controls a Unit in various command sources and monitors for fault conditions.

Use when:

- You want to consolidate status from groups of equipment. These statuses include:
 - Alarm Status
 - Alarm Priority
 - Command Source
 - Configuration Errors
- You want to manage and to following functions for a group of equipment, with a “global” set of commands:
 - Command Source
 - Alarm Acknowledge
 - Alarm Reset
- You want to apply permissive conditions to a group of equipment.
- You want to shut down groups of equipment based on a single alarm that occurs in any related equipment.
- You want to issue user-defined commands to equipment.

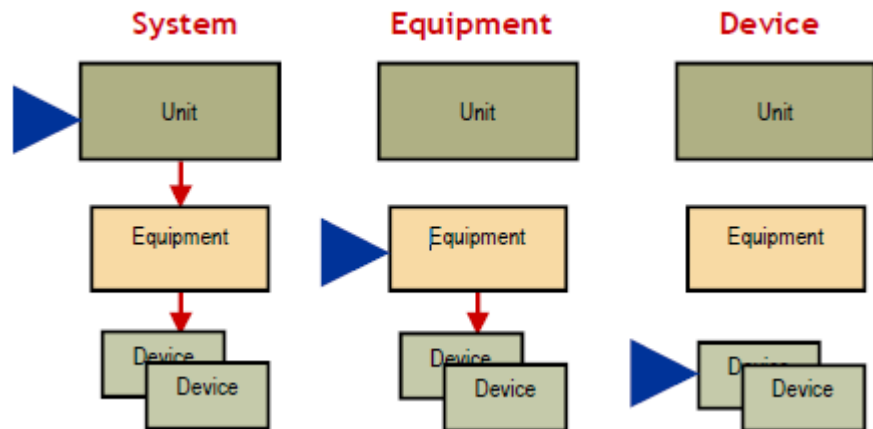
The raP_Opr_Unit object also:

- Provides an interface to parameter display, data entry, and configuration.
- Provides an interface to resultant (report) data display and configuration.
- Provides interface to Prompt Response and configuration

Functional Description

Command Source Management

Allows you to interact with the system at various levels.



Use the PCMDSRC PlantPax® instruction to manage the command source (owner) of an instruction or control strategy. For more information, see PlantPax Process Control Instructions, publication [PROCES-RM215](#).

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 Files

Controller File The raP_Opr_Unit_5.30.**00**_A0I.L5X or (raP_Opr_Unit_5.20.**00**_A0I.L5X) Add-On Instruction must be imported into the controller project to be used in the controller configuration. The maintenance release number (boldfaced) can change as maintenance revisions are created.

Visualization Files

See [Library Versions on page 15](#) for general information on visualization files.

Operations

Command Sources

The raP_Opr_Area instruction has no commands or outputs that are intended to control equipment and therefore does not have any selection of active command source.

Program Structure

The raP_Opr_Unit Instruction may be implemented using a program as a container (recommended). The following table outlines suggested program structure and routine naming:

Routine	Description
Object Name	Contains raP_Opr_Unit instance, external function instances (Interlock, Permissive, Extended Alarms), and routine calls.
AlarmsSuppress	Contains raP_Opr_Unit alarm suppression logic.
Interlocks	Contains raP_Opr_Unit interlock mapping from interlock conditions to _Intlk block.
Parameters	Contains raP_Opr_Unit parameter mapping to and from Parameter blocks (_ParRpt (Enum, Integer, Real, String)) to raP_Opr_Unit instance.
Permissives	Contains raP_Opr_Unit permissive mapping from permissive conditions to _Perm block.
Group Command Permissives	Contains raP_Opr_Unit group commands (1-4) permissive mapping from permissive conditions to _Perm block.
Reports	Contains raP_Opr_Unit report mapping to and from Parameter blocks (_ParRpt (Enum, Integer, Real, String)) to raP_Opr_Unit instance.
ExtddAlarms	Contains raP_Opr_Unit instances of external alarm instances and trigger logic.

IMPORTANT The raP_Opr_Unit Instruction may be implemented without the program structure that is defined in the previous table; this is provided as an example.

Alarms

The raP_Opr_Unit Instruction uses the following alarms, which are implemented by using Tag Based Alarms.

Alarm	Alarm Name	Description
E-Stop trip	Alm_EStopTrip	Raised when an emergency stop condition triggers a change in state of the Unit.
Group Command 1 Fail	Alm_GroupCmd1Fail	Raised when the defined Group Command 1 fails to execute on the Unit.
Group Command 2 Fail	Alm_GroupCmd2Fail	Raised when the defined Group Command 2 fails to execute on the Unit.
Group Command 3 Fail	Alm_GroupCmd3Fail	Raised when the defined Group Command 3 fails to execute on the Unit.
Group Command 4 Fail	Alm_GroupCmd4Fail	Raised when the defined Group Command 3 fails to execute on the Unit.
Interlock trip	Alm_IntlkTrip	Raised when an interlock condition triggers a change in state of the Unit.
S-Stop trip	Alm_SStopTrip	Raised when a software stop condition triggers a change in state of the Unit.

Virtualization

The raP_Opr_Unit Instruction has no Virtualization capability.

Execution

The handling of instruction execution conditions.

Condition	Description
EnableIn False (False Rung)	Handle processing for EnableIn False (False Rung) the same as if the Area were Disabled by Command. The Area outputs are de-energized and the Area is shown as Disabled on the HMI.
Powerup (Pre-scan, First Scan)	Handles processing of modes and alarms on Pre-scan and Powerup. On Powerup, the Area is treated as if it were Commanded to reset all program and operator commands
Postscan (SFC Transition)	No SFC Postscan logic is provided.

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

Local Message

The object raP_Opr_Unit utilizes local message display elements to display Step Names, Material Names, and Summary information. A default local message file is provided for each information type. This default local message file populates the local message display elements from tags in the controller. For Step Names and Material names, these are the same controller tags that are used in previous versions of the library. The difference is that 512 messages are available, rather than the 99 messages in the previous version. To upgrade from previous versions, developers must add the local message file to the project and set the @Navigation property of the specified tag to the Local Message file name (see the following table).

Information	Default Local Message File	File Name Reference	Default Controller Data
Material Name	SystemMaterialNames	Sts_eMtrl.@Navigation	System.Enum.Materials[x].@Description
Step Description	SystemStepDescriptions	Sts_eStep.@Navigation	System.Enum.Step_Desc[x].@Description
Summary Information	SystemSummary	Sts_eSummary.@Navigation	System.Enum.Summary_Desc[1].@Description

You may add customized local messages for individual objects by creating a new local message file and populating the file with the customized strings or tag references. Then set the @Navigation property of the specified tag to the name of the new custom file.

FactoryTalk Optix Local Message

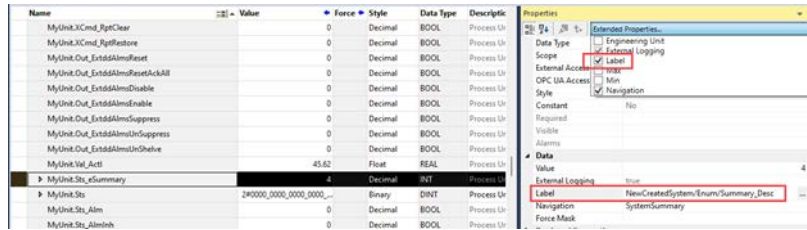
The object raP_Opr_Unit uses the @Label property of the specified tag to input the path for displaying Step Names, Material Names, and Summary information. FactoryTalk® Optix™ does not require Local Message files. For Step Names, Material Names, and Summary, these are the same controller tags that are used with FactoryTalk® View SE. FactoryTalk Optix directly retrieves Controller Data using the path that is specified in the @Label property of Logix Designer. You must set the path information into @Label property of the specified tag to a Controller Data Path (see the following table).

Information	Reference	Customized Controller Data Path	Default Controller Data
Material Name	Sts_eMtrl@Label	System/Enum/Materials	System.Enum.Materials[x].@Description
Step Description	Sts_eStep@Label	System/Enum/Step_Desc	System.Enum.Step_Desc[x].@Description
Summary Information	Sts_eSummary@Label	System/Enum/Summary_Desc	System.Enum.Summary_Desc[1].@Description

You may add Customized Controller Data for individual objects by creating a new tag member with the Data Type "raP_UDT_Opr_System" in Logix Designer.

Name	Alias For	Base Tag	Data Type	Description	External Access
System			raP_UDT_Opr_System	System Global Structure	Read/Write
System.Enum			raP_UDT_Opr_System_Enum	System Global Structure Enumerations	Read/Write
System.Enum.Step_Desc			BOOL[512]	System Global Structure Step Descriptions	Read/Write
System.Enum.Materials			BOOL[512]	System Global Structure Material Names	Read/Write
System.Enum.Summary_Desc			BOOL[512]	System Global Structure Summary Descriptions	Read/Write
System.Proj			raP_UDT_Opr_System_Proj	System Global Structure Project Settings	Read/Write
System.Sts			raP_UDT_Opr_System_Status	System Global Structure Status	Read/Write
NewCreatedSystem			raP_UDT_Opr_System	System Global Structure	Read/Write

Then set the @Label property of the specified tag to the Customized Controller Data Path and import the tag with the customized extended properties into FactoryTalk Optix.



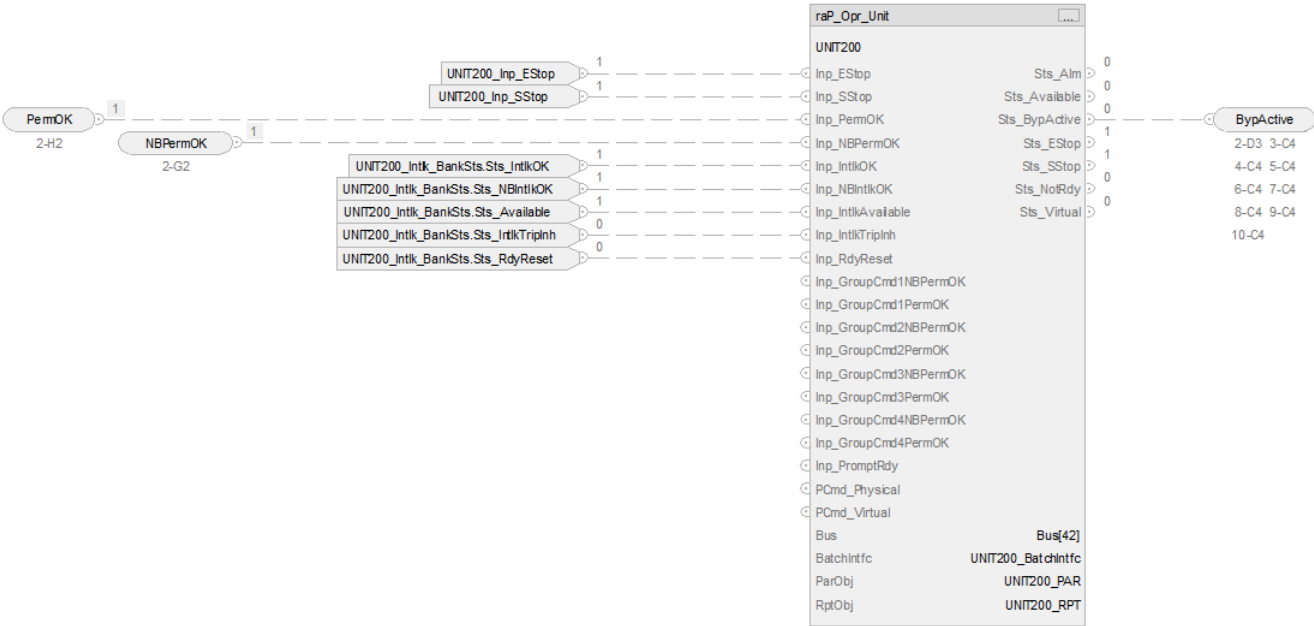
Tag Extended Properties and Default Alarm Settings

Common	
raP_Opr_Unit.@Description	"Unit"
raP_Opr_Unit.@Area	"Area01"
raP_Opr_Unit.@Instruction	"raP_Opr_Unit"
raP_Opr_Unit.@Label	"Unit"
raP_Opr_Unit.@Library	"raP-5_30"
raP_Opr_Unit.@URL	"n/a"
raP_Opr_Unit.Cfg_HasMoreObj.@Navigation	" "
General	
raP_Opr_Unit.Sts.0.@Description	"State 0"
raP_Opr_Unit.Sts.1.@Description	"State 1"
raP_Opr_Unit.Sts.2.@Description	"State 2"
raP_Opr_Unit.Sts.3.@Description	"State 3"
raP_Opr_Unit.Sts.ExtddAlms.@Label	"Extended alarm"
raP_Opr_Unit.XCmd.0.@Description	"Group Command 0"
raP_Opr_Unit.XCmd.1.@Description	"n/a"
raP_Opr_Unit.XCmd.2.@Description	"n/a"
raP_Opr_Unit.XCmd.3.@Description	"n/a"
raP_Opr_Unit.Val.Actl.@EngineeringUnit	"Kg"

Alarms		Alarm Default Message	Severity
raP_Opr_Unit.Sts.EStopTrip.@Label	"Emergency stop"	"/*S:0 %.@Description*/: Emergency stop	750
raP_Opr_Unit.Sts.GroupCmd1Fail.@Label	"Group Command 1 Failed"	"/*S:0 %.@Description*/: Group Command 1 Failed	500
raP_Opr_Unit.Sts.GroupCmd2Fail.@Label	"Group Command 2 Failed"	"/*S:0 %.@Description*/: Group Command 2 Failed	500
raP_Opr_Unit.Sts.GroupCmd3Fail.@Label	"Group Command 3 Failed"	"/*S:0 %.@Description*/: Group Command 3 Failed	500
raP_Opr_Unit.Sts.GroupCmd4Fail.@Label	"Group Command 4 Failed"	"/*S:0 %.@Description*/: Group Command 4 Failed	500
raP_Opr_Unit.Sts.IntlkTrip.@Label	"Interlock trip"	"/*S:0 %.@Description*/: Interlock trip	500
raP_Opr_Unit.Sts.SStopTrip.@Label	"Software stop"	"/*S:0 %.@Description*/: Software stop	750

Programming Example

The example in the Functional Description section shows the basic use of the raP_Opr_Unit Add-On Instruction. Typically, the raP_Opr_Unit instruction is used in association with a Bus-resident entity. The following example shows a unit instruction that is associated with a Bus referenced entity. The unit is also connected to two inputs, emergency stop and software stop, that can trigger two alarms. Units also allow for connections to permissives and interlocks. The unit object is typically used in an S88 application but can be applied to suit numerous hierarchy layouts. The unit allows for optional connections to parameters, reports, and FTBatch interface objects.



Generic Equipment Module (raP_Opr_EMGen)

An equipment module is a functional group of equipment that can conduct a finite number of specific minor processing activities. An equipment module is typically centered around a piece of process equipment (a weigh tank, a process heater, a scrubber, and so forth). This term applies to both the physical equipment and the equipment entity.



For the object and visualization parameters, see PlantPAx Process Objects, publication [PROCES-RD200](#), and PlantPAx Visualization Files, publication [PROCES-RD201](#).

Guidelines

The raP_Opr_EMGen (Generic Equipment Module) object controls an Equipment Module in various command sources and monitors for fault conditions.

Use when:

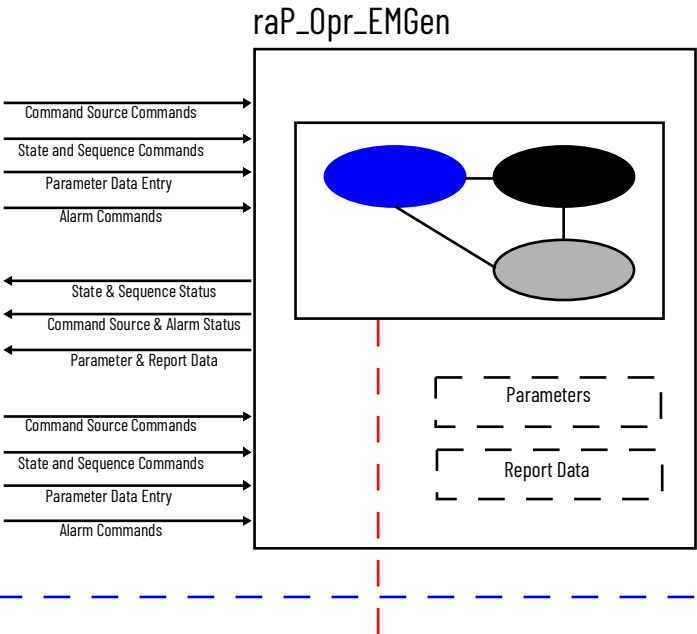
- You want to group equipment, and you want to apply a custom state model
- You want to provide the following for a group of equipment
 - Apply a mode model to the equipment group
 - Definable Commands and states
 - Apply interlocks and/or permissives to the group of equipment
 - Parameter that define the behavior of the group of equipment
 - Report / Resultant data from the group of equipment
 - A faceplate that allows monitoring / control of the equipment grouping
 - Alarm if any device fails
 - Monitor step (description), and allow forcing of steps in maintenance command source
 - Allow configurable alarms for certain process / equipment failure conditions

Functional Description

Program

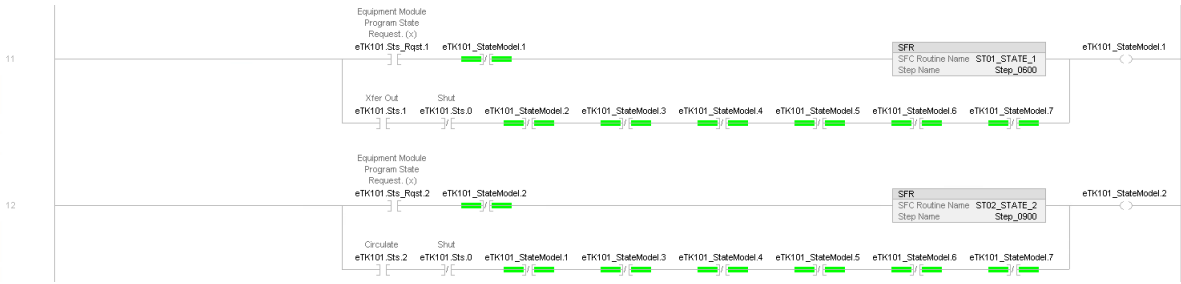
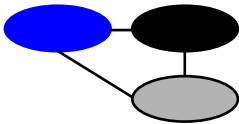
Dispatch

Contains raP_Opr_EMGen instruction and any external instructions required.



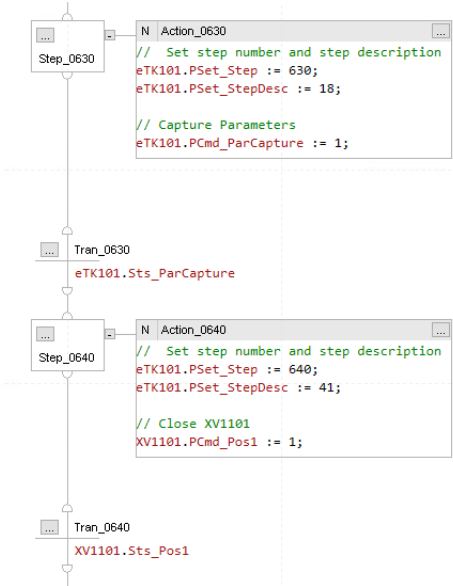
StateModel

Contains your state model (if state model is implemented external to raP_Opr_EMGen)



STxx_<State> Routines

Contains your logic that sequences and coordinates devices (implement states as required)



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 raP_Opr_EMGen_5.30.**00**_A01.L5X or (raP_Opr_EMGen_5.20.**00**_A01.L5X) Add-On Instruction must be imported into the controller project to be used in the controller configuration. The maintenance release number (boldfaced) can change as maintenance revisions are created.

Visualization Files

See [Library Versions on page 15](#) for general information on visualization files.

Operations

The primary operations of raP_Opr_EMGen (Generic Equipment Module) are to:

- Provides user-defined states, and commands
- Allow monitoring of sequence Step, and display sequence Status.
- Monitor permissive conditions to help prevent Equipment Module operation.
- Monitor interlock conditions to help prevent Equipment Module operation or create failure condition.
- Provide the ability to force steps (maintenance)
- Monitor various Equipment Module failure conditions, and produce alarms.
- Operate in maintenance, program, and operator command source.
- Provide an “available” status for use by automation logic, to indicate that the Equipment Module is available for operation.
- Provide a propagation mechanism to allow the Equipment Module to publish status to and receive status from a group of equipment.
- Provides an interface to parameter display, data entry, and configuration.
- Provides an interface to resultant (report) data display and configuration.
- Allows configurable state effect of Alarm and Permissive
- Provides interface to Prompt Response and configuration

Command Sources

The raP_Opr_EMGen (Generic Equipment Module) uses the standard command source operations that are implemented using an embedded PCMDSRC instruction. See PlantPAx Process Control Instructions, publication [PROCES-RM215](#) for more information.

State Model

The raP_Opr_EMGen (Generic Equipment Module) Add-On Instruction allows the creation of a customized state model for a particular instance.

Depending on your requirements, you may choose to write your own State module logic and make the appropriate connections to the raP_Opr_EMGen, or you may choose to use one of the provided raP_Opr_VSM (Variable State Module) configurations (S88, NUMUR, PackML, Equipment, and Generic), or create your own using this provided Add-On Instruction. You can then make the appropriate connections to the raP_Opr_EMGen. Each instance of the raP_Opr_EMGen needs an instance of the raP_Opr_VSM Instruction.

The raP_Opr_EMGen (Generic Equipment Module) provides up to 32 state commands (PCmd) and 32 state status's (Sts), which may be used when creating a custom state model.

Program Structure

The raP_Opr_EMGen (Generic Equipment Module) may be implemented using a program as a container (recommended). The following table outlines suggested program structure and routine naming:

Routine	Description
Dispatch	Contains raP_Opr_EMGen instance, external function instances (Interlock, Permissive, Associated Device), and routine calls.
AlarmsSuppress	Contains raP_Opr_EMGen alarm suppression logic.
Interlocks	Contains raP_Opr_EMGen interlock mapping from interlock conditions to _Intlk block.
Parameters	Contains raP_Opr_EMGen parameter mapping to and from Parameter blocks (_ParRpt (Enum, Integer, Real, String)) to raP_Opr_EMGen instance.
Permissives	Contains raP_Opr_EMGen permissive mapping from permissive conditions to _Perm block.
Reports	Contains raP_Opr_EMGen report mapping to and from Parameter blocks (_ParRpt (Enum, Integer, Real, String)) to raP_Opr_EMGen instance.
_StateModel	Contains raP_Opr_EMGen state module program logic.
ExtddAlarms	Contains raP_Opr_EMGen instances of external alarm instances and trigger logic.
St<xx>_<StateDesc>	Contains raP_Opr_EMGen state logic.

IMPORTANT The raP_Opr_EMGen (Generic Equipment Module) may be implemented without the program structure that is defined in the preceding table; this is provided as an example.

Alarms

The raP_Opr_EMGen Instruction uses the following alarms, which are implemented by using Tag Based Alarms.

Alarm	Alarm Name	Description
Device alarms	Alm_DvcAlms	Raised when a device within the Equipment Module has an alarm.
Interlock trip	Alm_IntlkTrip	Raised when an interlock condition triggers a change in state of the Equipment Module.
Report data	Alm_RptData	Raised when new report data are available.

Virtualization

The raP_Opr_EMGen Instruction has no Virtualization capability.

Execution

Condition	Description
EnableIn False (False Rung)	Handle processing for EnableIn False (False Rung) the same as if the Equipment Module were Disabled by Command. The Equipment Module outputs are de-energized and the Equipment Module is shown as Disabled on the HMI.
Powerup (Pre-scan, First Scan)	Handles processing of command sources and alarms on Pre-scan and Powerup. On Powerup, the Equipment Module is treated as if it were Commanded to Reset all Program and Operator commands.
Postscan (SFC Transition)	No SFC Postscan logic is provided.

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



ATTENTION: Disabling the raP_Opr_EMGen Add-On Instruction causes Equipment Module outputs to become de-energized.

Local Message

The object raP_Opr_EMGen utilizes local message display elements to display Step Names, Material Names, and Summary information. A default local message file is provided for each information type. This default local message file populates the local message display elements from tags in the controller. For Step Names and Material names, these are the same controller tags that are used in previous versions of the library. The difference is that 512 messages are available, rather than the 99 messages in the previous version. To upgrade from previous versions, developers must add the local message file to the project and set the @Navigation property of the specified tag to the Local Message file name (see the following table).

Information	Default Local Message File	File Name Reference	Default Controller Data
Material Name	SystemMaterialNames	Sts_eMtrl.@Navigation	System.Enum.Materials[x].@Description
Step Description	SystemStepDescriptions	Sts_eStep.@Navigation	System.Enum.Step_Desc[x].@Description
Summary Information	SystemSummary	Sts_eSummary.@Navigation	System.Enum.Summary_Desc[1].@Description

Users may add customized local messages for individual objects by creating a new local message file and populating the file with the customized strings or tag references. Then set the @Navigation property of the specified tag to the name of the new custom file.

FactoryTalk Optix Local Message

The object raP_Opr_Unit uses the @Label property of the specified tag to input the path for displaying Step Names, Material Names, and Summary information. FactoryTalk Optix does not require Local Message files. For Step Names, Material Names, and Summary, these are the same controller tags that are used with FactoryTalk® View SE. FactoryTalk Optix directly retrieves Controller Data using the path that is specified in the @Label property of Logix Designer. Users must set the path information into @Label property of the specified tag to a Controller Data Path (see the following table).

Information	Reference	Customized Controller Data Path	Default Controller Data
Material Name	Sts_eMtrl@Label	System/Enum/Materials	System.Enum.Materials[x].@Description
Step Description	Sts_eStep@Label	System/Enum/Step_Desc	System.Enum.Step_Desc[x].@Description
Summary Information	Sts_eSummary@Label	System/Enum/Summary_Desc	System.Enum.Summary_Desc[1].@Description

You may add Customized Controller Data for individual objects by creating a new tag member with the Data Type "raP_UDT_Opr_System" in Logix Designer.

Name	Alias For	Base Tag	Data Type	Description	External Access
System			raP_UDT_Opr_System	System Global Structure	Read/Write
System.Enum			raP_UDT_Opr_System_Enum	System Global Structure Enumerations	Read/Write
System.Enum.Step_Desc			BOOL[512]	System Global Structure Step Descriptions	Read/Write
System.Enum.Materials			BOOL[512]	System Global Structure Material Names	Read/Write
System.Enum.Summary_Desc			BOOL[512]	System Global Structure Summary Descriptions	Read/Write
System.Proj			raP_UDT_Opr_System_Proj	System Global Structure Project Settings	Read/Write
System.Sts			raP_UDT_Opr_System_Status	System Global Structure Status	Read/Write
NewCreatedSystem			raP_UDT_Opr_System	System Global Structure	Read/Write

Then set the @Label property of the specified tag to the Customized Controller Data Path and import the tag with the customized extended properties into FactoryTalk Optix.

Name	Value	Force	Style	Data Type	Description
MyLink.XCmd_RptClear	0	Decimal	BOOL	Process U-	
MyLink.XCmd_RptFeature	0	Decimal	BOOL	Process U-	
MyLink.Out_ExtddAlarmReset	0	Decimal	BOOL	Process U-	
MyLink.Out_ExtddAlarmAckAll	0	Decimal	BOOL	Process U-	
MyLink.Out_ExtddAlarmDisable	0	Decimal	BOOL	Process U-	
MyLink.Out_ExtddAlarmEnable	0	Decimal	BOOL	Process U-	
MyLink.Out_ExtddAlarmSuppress	0	Decimal	BOOL	Process U-	
MyLink.Out_ExtddAlarmInShelve	0	Decimal	BOOL	Process U-	
MyLink.Val_Act1	45.62	Float	REAL	Process U-	
MyLink.Sts_eSummary	4	Decimal	DINT	Process U-	
MyLink.Sts	240000_0000_0000_0000_...	Binary	DINT	Process U-	
MyLink.Sts_Alarm	0	Decimal	BOOL	Process U-	
MyLink.Sts_AlarmInb	0	Decimal	BOOL	Process U-	

Properties
Extended Properties...
Data Type Engineering Unit
Scope External Logging
External Access Label
OPC UA Access Min
Style Navigation
Constant No
Required
Visible
Alarms
Data Value 4
External Logging true
Label NewCreatedSystem/Enum/Summary_Desc
Navigation SystemSummary
Force Mask

Tag Extended Properties and Default Alarm Settings

Common	
raP_Opr_EMGen.@Description	"Generic Equipment Module"
raP_Opr_EMGen.@Area	"Area01"
raP_Opr_EMGen.@Instruction	"raP_Opr_EMGen"
raP_Opr_EMGen.@Label	"Equipment Module"
raP_Opr_EMGen.@Library	"raP-5_30"
raP_Opr_EMGen.@URL	" "
raP_Opr_EMGen.Cfg_HasMoreObj.@Navigation	" "
raP_Opr_EMGen.Sts_eStep.@Navigation	"SystemStepDescriptions"
raP_Opr_EMGen.Sts_eSummary.@Navigation	"SystemSummary"
raP_Opr_EMGen.Cfg_HasDvcAlmsObjNav@Navigation	" "
raP_Opr_EMGen.Ref_SMCfgPath@Navigation	" "

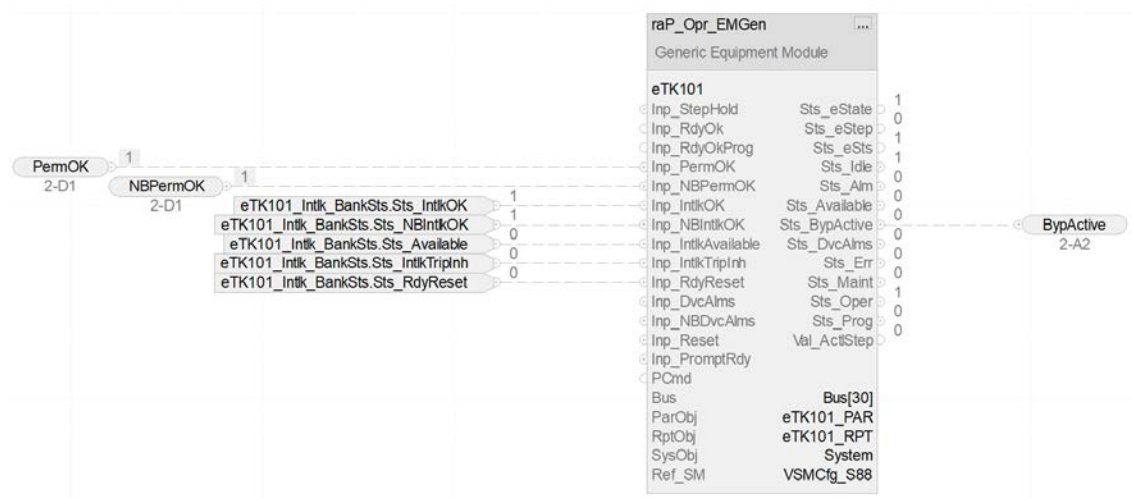
General	
raP_Opr_EMGen.Cfg_HasDetailDisplay.@Navigation	" "
raP_Opr_EMGen.Sts.0.@Description	"State 1"
raP_Opr_EMGen.Sts.1.@Description	"State 2"
raP_Opr_EMGen.Sts.2.@Description	"State 3"
raP_Opr_EMGen.Sts.3.@Description	"State 4"
raP_Opr_EMGen.Sts.4.@Description	"State 5"
raP_Opr_EMGen.Sts.5.@Description	"State 6"
raP_Opr_EMGen.Sts.6.@Description	"State 7"
raP_Opr_EMGen.Sts.7.@Description	"State 8"
raP_Opr_EMGen.Sts.8.@Description	"State 9"
raP_Opr_EMGen.Sts.9.@Description	"State 10"
raP_Opr_EMGen.Sts.10.@Description	"State 11"
raP_Opr_EMGen.Sts.11.@Description	"State 12"
raP_Opr_EMGen.Sts.12.@Description	"State 13"
raP_Opr_EMGen.Sts.13.@Description	"State 14"
raP_Opr_EMGen.Sts.14.@Description	"State 15"
raP_Opr_EMGen.Sts.15.@Description	"State 16"
raP_Opr_EMGen.Sts.16.@Description	"State 17"
raP_Opr_EMGen.Sts.17.@Description	"State 18"
raP_Opr_EMGen.Sts.18.@Description	"State 19"
raP_Opr_EMGen.Sts.19.@Description	"State 20"
raP_Opr_EMGen.Sts.20.@Description	"State 21"
raP_Opr_EMGen.Sts.21.@Description	"State 22"
raP_Opr_EMGen.Sts.22.@Description	"State 23"
raP_Opr_EMGen.Sts.23.@Description	"State 24"
raP_Opr_EMGen.Sts.24.@Description	"State 25"
raP_Opr_EMGen.Sts.25.@Description	"State 26"
raP_Opr_EMGen.Sts.26.@Description	"State 27"
raP_Opr_EMGen.Sts.27.@Description	"State 28"
raP_Opr_EMGen.Sts.28.@Description	"State 29"
raP_Opr_EMGen.Sts.29.@Description	"State 30"
raP_Opr_EMGen.Sts.30.@Description	"State 31"
raP_Opr_EMGen.Sts.31.@Description	"State 32"
raP_Opr_EMGen.Sts_ExtddAlms.@Label	"Extended alarm"
raP_Opr_EMGen.XCmd.0.@Description	"Command 1"
raP_Opr_EMGen.XCmd.1.@Description	"n/a"
raP_Opr_EMGen.XCmd.2.@Description	"n/a"
raP_Opr_EMGen.XCmd.3.@Description	"n/a"

General	
raP_Opr_EMGen.XCmd.4.@Description	"n/a"
raP_Opr_EMGen.XCmd.5.@Description	"n/a"
raP_Opr_EMGen.XCmd.6.@Description	"n/a"
raP_Opr_EMGen.XCmd.7.@Description	"n/a"
raP_Opr_EMGen.XCmd.8.@Description	"n/a"
raP_Opr_EMGen.XCmd.9.@Description	"n/a"
raP_Opr_EMGen.XCmd.10.@Description	"n/a"
raP_Opr_EMGen.XCmd.11.@Description	"n/a"
raP_Opr_EMGen.XCmd.12.@Description	"n/a"
raP_Opr_EMGen.XCmd.13.@Description	"n/a"
raP_Opr_EMGen.XCmd.14.@Description	"n/a"
raP_Opr_EMGen.XCmd.15.@Description	"n/a"
raP_Opr_EMGen.XCmd.16.@Description	"n/a"
raP_Opr_EMGen.XCmd.17.@Description	"n/a"
raP_Opr_EMGen.XCmd.18.@Description	"n/a"
raP_Opr_EMGen.XCmd.19.@Description	"n/a"
raP_Opr_EMGen.XCmd.20.@Description	"n/a"
raP_Opr_EMGen.XCmd.21.@Description	"n/a"
raP_Opr_EMGen.XCmd.22.@Description	"n/a"
raP_Opr_EMGen.XCmd.23.@Description	"n/a"
raP_Opr_EMGen.XCmd.24.@Description	"n/a"
raP_Opr_EMGen.XCmd.25.@Description	"n/a"
raP_Opr_EMGen.XCmd.26.@Description	"n/a"
raP_Opr_EMGen.XCmd.27.@Description	"n/a"
raP_Opr_EMGen.XCmd.28.@Description	"n/a"
raP_Opr_EMGen.XCmd.29.@Description	"n/a"
raP_Opr_EMGen.XCmd.30.@Description	"n/a"
raP_Opr_EMGen.XCmd.31.@Description	"n/a"

Alarms		Alarm Default Message	Severity
raP_Opr_EMGen.Sts_DvcAlms.@Label	"Device alarm"	"/*S:0 %.@Description*/: Device alarm	500
raP_Opr_EMGen.Sts_IntlkTrip.@Label	"Interlock trip"	"/*S:0 %.@Description*/: Interlock trip	500
raP_Opr_EMGen.Sts_RptData.@Label	"Report data not collected"	"/*S:0 %.@Description*/: Report data	500

Programming Example

The example in the Functional Description section shows the basic use of the raP_Opr_EMGen Add-On Instruction. Typically, the raP_Opr_EMGen instruction is used in association with a Bus-resident entity. The following shows a generic equipment module instruction that is associated with a Bus referenced entity. The generic equipment module also allows for connections to permissives, interlocks, and a System tag. The generic equipment module is typically used in an S88 application but can be applied to suit numerous hierarchy layouts. The generic equipment module allows for optional connections to parameter and report interface objects.



Notes:

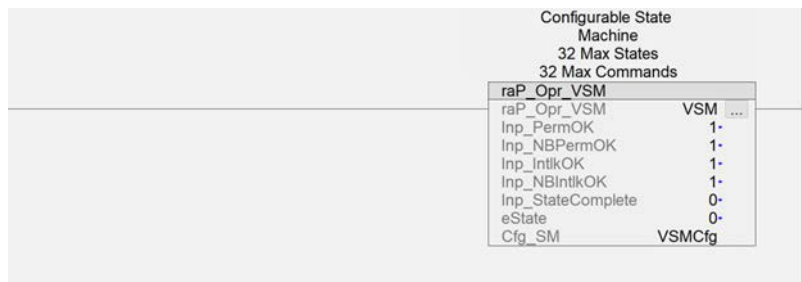
Variable State Machine (raP_Opr_VSM)

IMPORTANT raP_Opr_VSM is available in Process Library 5.30 and later.

The Variable State Machine (raP_Opr_VSM) allows you to create your own state machine with up to 32 states and 32 commands.

The state machine configuration is contained in a separate data structure instance (raP_UDT_Opr_VSMcfg). This is so that multiple instances of the state machine can use the same state machine configuration. This facilitates uniformity and ease of editing and troubleshooting.

Functional Description



Permissive

The permissive OK inputs determine the ability to be commanded from the configured idle state. These inputs do not influence state done (Inp_StateDN) behavior. If the permissive is not OK, then the state machine cannot be commanded to leave the configured idle state. All valid commands are inhibited for that state and the VSM Sts_NrdyPerm=1. The Sts_CmdIPInh shows the commands that the configuration would normally allow but are currently inhibited.

Name	Type	Default	Description
Inp_PermOK	BOOL	Default=0	Input for bypassable permissive.
Inp_NBPermOK	BOOL	Default=0	Input for non-bypassable permissives.
Sts_NrdyPerm	BOOL		Status to indicate that the permissive is not OK when the state machine is in idle.
Sts_CmdIPInh	DINT		Bit mapped status that indicates which valid commands are being inhibited by either an Interlock or a permissive.

Interlock

Optionally, you can issue a command to the state machine when an interlock is not OK. When this configuration causes a command to be issued, then the `Sts_IntlkTrip=1` for one scan when that command is issued. The state machine will not trip when an interlock is not OK while in the configured idle state.

Optionally, you can block commands from being issued to the state machine when an interlock is not OK. When this configuration is causing commands to be inhibited, then the `Sts_NrdyIntlk=1`. The `Sts_CmdIPlnh` shows the commands that the configuration would normally allow but are currently inhibited.

Name	type	Default	Description
Inp_IntlkOK	BOOL	Default=0	Input for bypassable interlocks.
Inp_NBIIntlkOK	BOOL	Default=0	Input for non-bypassable interlocks.
Cfg_IntlkAsPerm	BOOL	Default=0	Configuration to use the interlock status as a permissive and an interlock.
Cfg_CmdIntlkAsEdge	BOOL	Default=0	Configuration to treat the interlock as a leading edge triggered signal.
Set_IntlkCmd ⁽¹⁾	SINT	Default= -1 (no command)	Command number to issue to the state machine if the interlock is not OK.
Sts_IntlkTrip	BOOL		Status asserted if the interlock caused a valid command to be issued to the state machine as set in <code>Set_IntlkCmd</code> . When asserted this status is high (1) for a single scan.
Sts_NrdyIntlk	BOOL		Status to indicate that the interlock is not OK and is inhibiting valid commands.
Sts_CmdIPlnh	DINT		Bit mapped status that indicates which valid commands are being inhibited by either an Interlock or a permissive.
Cfg.Cfg_InhCmdOnIntlk	DINT	Default=0	Bit mapped configuration to inhibit individual commands when an interlock is not OK.

(1) When the VSM is used within the `raP_Opr_EMGen` object, the `Set_IntlkCmd` is set by the `EM.Cfg_IntlkTripState` value, which is defined from the equipment module advanced faceplate.

Command Source

The `raP_Opr_VSM` has no internal `CmdSrc` instruction. An external `CmdSrc` can be used to communicate the `CmdSrc` state to the `raP_Opr_VSM` via the `Inp_eSrc` input. The `raP_Opr_VSM` will then update its command enables and options that are associated with the different `CmdSrc` states.

`raP_Opr_VSM.Inp_eSrc := CmdSrc.Sts_eSrc`

On startup, reset or initialization if the `raP_Opr_VSM` detects the absence of a `CmdSrc` via this input then it enables Operator and Program states internally. This enables functionality for both Operator and Program inputs as well as the associated status.

Command Source	Description
Operator/Maintenance	OCMD - DINT. One Shot Latch the bit corresponding to the command. Ordy_CmdEnable - DINT. Command is enabled when corresponding bit is on (1).
Program	PCMD - DINT. One Shot Latch the bit corresponding to the command. PSet_StateNum - SINT. Set the state number of the desired state.
External	XCMD - DINT. One Shot Latch the bit corresponding to the command. XRdy_CmdEnable - DINT. Command is enabled when corresponding bit is on (1).
Override	Inp_OvrCmd - DINT. One Shot Latch the bit corresponding to the command. Inp_OvrStateNum - SINT. Set the state number of the desired state.
OoS	When Out of Service (or the instruction is scanned false) the <code>raP_Opr_VSM</code> will continuously initialize. Not allowing any commands or state changes. The state machine is held in state zero (0).
Hand	Inp_HandStateNum - SINT. Set the state number of the desired state.

State Complete

Name	Description
Inp_StateComplete	Input for external logic to tell the state machine that the current state is complete.
Cfg_StateCompleteAsEdge	Configuration to require the Inp_StateComplete to go low before it is again recognized.

Idle State

Name	type	Default	Description
Cfg_IdleStateNum	SINT	0	Configured Idle State Number (0...31). VSM Initializes to the configured idle state. Permissive are checked while in the idle state and the interlocks will not trip while in the idle state. The return address will always be updated while in idle.

IMPORTANT When the VSM is used within the raP_Opr_EMGen object, the Cfg_IdleStateNum is assigned the same value as the EM.Cfg_IdleState configuration automatically.

Auxiliary Commands

Name	Description
Inp_CmdAux1:	1 = Assert command number that is assigned to Set_CmdAux1
Inp_CmdAux2	1 = Assert command number that is assigned to Set_CmdAux2
Cfg_CmdAux1AsEdge	1 = Inp_CmdAux1 as Edge
Cfg_CmdAux2AsEdge	1 = Inp_CmdAux2 As Edge
Set_CmdAux1 ⁽¹⁾	Command Number to issue when Inp_AuxCmd1 = 1
Set_CmdAux2	Command Number to issue when Inp_AuxCmd2 = 1.

(1) When the VSM is used within the raP_Opr_EMGen object, the Set_CmdAux1 is set by the EM.Cfg_NrdyState value, which is defined from the equipment module advanced faceplate.

State Machine Configuration (raP_UDT_Opr_VSMCfg)

State Configuration	Type	Description
Cfg_Cmd4All	DINT	Cfg_Cmd4All[Command Number] = 1. Commands available in all states.
Cfg_DontUpdRetSt	DINT	Cfg_DontUpdRetSt.[State Number] = 1. Don't update the return address in this state.
Cfg_InhCmdOnIntlk	DINT	Cfg_InhCmdOnIntlk.[Command Number] = 1. Don't allow this command if the interlock is not OK.
Cfg_CmdMask	DINT[32]	Cfg_CmdMask[State Number] = 1. Allows that command in that state.
Cfg_CmdTarget	SINT[32]	Cfg_CmdTarget[Command Number] = Target State Number. Defines the destination state of a command.
Cfg_SCTarget	SINT[32]	Cfg_SCTarget[State Number] = Target State when Inp_StateComplete = 1.

Cfg_Cmd4All

Each bit of this DINT corresponds to a command in the state machine. When the corresponding bit is set, this command is available to all CmdSrc owners simultaneously when the command is enabled. This can be useful if a 'Stop' command should always be available to any CmdSrc owner when enabled.

Example:

Cfg_Cmd4All = 2# 0000 0000 0000 0000 0000 1010 0000 0010

Commands 1, 9, 11 are available to be asserted by Oper, Prog, External, and Maintenance command inputs when those commands are enabled.

Cfg_DontUpdRetSt

Each bit of this DINT corresponds to a state in the state machine. Internally to the state machine there is a register that maintains the number of the state as it proceeds through the states. An option for command targets is to use the state number stored in this register. When the corresponding bit is set this state number is not updated with the corresponding state when it is current. This can be useful if multiple states can receive a 'Hold' command. But upon 'Restart' it is desirable to return to the state from which it came instead of the last state encountered or a single specified state.

Example:

```
Cfg_DontUpdRetSt = 2# 0000 0000 0000 0000 0000 0000 1101 0000
```

States 4, 6, 7 will not update this register as it passes through them. If in one of these states and a command to transition to the state held in the register, then the state machine transitions to the last recorded state (not one of these). If a state machine has 'Starting' and 'Running' states but both can be interrupted by a 'Hold' command then it may be desirable to return to whichever state was interrupted. In this case, the corresponding bits for the 'Held' And 'Restarting' states would be set. Leaving the state number for either 'Starting' or 'Holding' in the register. Then the complete target for the 'Restarting' state would be the return state in the register.

Cfg_InhCmdOnIntlk

Each bit of this DINT corresponds to a command in the state machine. When the corresponding bit is set, this command is inhibited when normally it would be enabled by the configuration but the interlock is not OK. This is useful to inhibit 'Start' or 'Restart' commands if the interlock is not OK.

Example:

```
Cfg_InhCmdOnIntlk = 2# 0000 0000 0000 0000 0100 0001 0000 0001
```

Commands 0, 8, 14 are inhibited if the interlock is not OK.

Cfg_CmdMask

Each DINT element corresponds to a state in the state machine. Each bit of that element corresponds to a command in the state machine.

Example:

```
Cfg_CmdMask[3] = 2# 0000 0000 0000 0000 0000 0010 0011 1000
```

In state 3 commands 3, 4, 5, 9 are enabled.

Cfg_CmdTarget

Each SINT element corresponds to a command in the state machine. The numerical value of that element is the number of the state that is the target when this command is received.

Example:

```
Cfg_CmdTarget[2] = 5
```

When command 2 is received the target state is state number 5.

Cfg_SCTarget

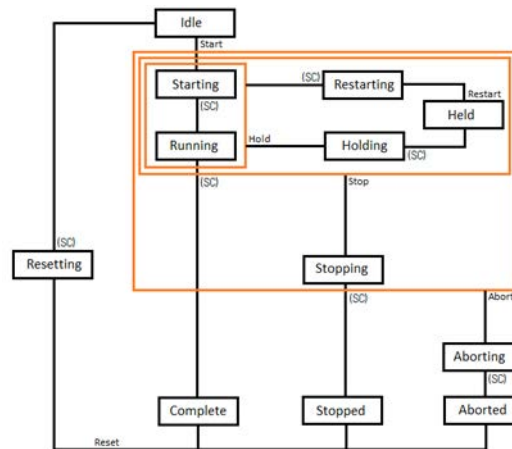
Each SINT element corresponds to a state in the state machine. The numerical value of that element is the number of the state that is the target when this state is complete.

Example:

```
Cfg_SCTarget[7] = 9
```

When state 7 is declared complete (by Inp_StateComplete=1) then the target state is state number 9.

Configuration Example



States	State Complete Target	Start ⁽¹⁾	Reset ⁽¹⁾	Restart ⁽¹⁾	Hold ⁽¹⁾	Stop ⁽¹⁾	Abort ⁽¹⁾
0 - Idle	0 Idle	X					
1 - Starting	2 Running				X	X	X
2 - Running	3 Complete				X	X	X
3 - Complete	3 Complete		X				
4 - Holding	5 Held					X	X
5 - Held	5 Held			X		X	X
6 - Restarting	-1 (last state)				X	X	X
7 - Stopping	8 Stopped						X
8 - Stopped	8 Stopped		X				
9 - Aborting	10 Aborted						
10 - Aborted	10 Aborted		X				
11 - Resetting	0 Idle						

(1) Command accepted in state.

IMPORTANT

Commands of higher precedence are assigned a higher bit number than commands of lower precedence. When multiple simultaneous commands are received the command with the highest precedence is the command that is executed.

Allowable Command Mask	State: Commands
Cfg_CmdMask[0] = 2# 0000 0000 0000 0000 0000 0000 0001	Idle: Start
Cfg_CmdMask[1] = 2# 0000 0000 0000 0000 0000 0000 0011 1000	Starting: Hold, Stop, Abort
Cfg_CmdMask[2] = 2# 0000 0000 0000 0000 0000 0000 0011 1000	Running: Hold, Stop, Abort
Cfg_CmdMask[3] = 2# 0000 0000 0000 0000 0000 0000 0000 0010	Complete: Reset
Cfg_CmdMask[4] = 2# 0000 0000 0000 0000 0000 0000 0011 0000	Holding: Stop, Abort
Cfg_CmdMask[5] = 2# 0000 0000 0000 0000 0000 0000 0011 0100	Held: Restart, Stop, Abort
Cfg_CmdMask[6] = 2# 0000 0000 0000 0000 0000 0000 0011 1000	Restarting: Hold, Stop, Abort
Cfg_CmdMask[7] = 2# 0000 0000 0000 0000 0000 0000 0010 0000	Stopping: Abort
Cfg_CmdMask[8] = 2# 0000 0000 0000 0000 0000 0000 0000 0010	Stopped: Reset
Cfg_CmdMask[9] = 2# 0000 0000 0000 0000 0000 0000 0000 0000	Aborting: *None
Cfg_CmdMask[10] = 2# 0000 0000 0000 0000 0000 0000 0000 0010	Aborted: Reset
Cfg_CmdMask[11] = 2# 0000 0000 0000 0000 0000 0000 0000 0000	Resetting: *None

State Target of Command	Command: State
Cfg_CmdTarget[0] = 1	Start: Starting
Cfg_CmdTarget[1] = 11	Reset: Resetting
Cfg_CmdTarget[2] = 6	Restart: Restarting

State Target of Command	Command: State
Cfg_CmdTarget[3] = 4	Hold: Holding
Cfg_CmdTarget[4] = 7	Stop: Stopping
Cfg_CmdTarget[5] = 9	Abort: Aborting

State Target of This State Complete	State: State Complete Target
Cfg_SCTarget[0] = 0	Idle: Idle
Cfg_SCTarget[1] = 2	Starting: Running
Cfg_SCTarget[2] = 3	Running: Complete
Cfg_SCTarget[3] = 3	Complete: Complete
Cfg_SCTarget[4] = 5	Holding: Held
Cfg_SCTarget[5] = 5	Held: Held
Cfg_SCTarget[6] = -1	Restarting: *Return State (-1)
Cfg_SCTarget[7] = 8	Stopping: Stopped
Cfg_SCTarget[8] = 8	Stopped: Stopped
Cfg_SCTarget[9] = 10	Aborting: Aborted
Cfg_SCTarget[10] = 10	Aborted: Aborted
Cfg_SCTarget[11] = 0	Resetting: Idle

Cfg_Cmd4All = 2# 0000 0000 0000 0000 0000 0000 0010 0000

This makes the 'Abort' command (5) available to all CmdSrc owners regardless of the current CmdSrc state when enabled.

Cfg_DontUpdRetSt = 2# 0000 0000 0000 0000 0000 0000 0111 0000

Don't record the state numbers for the 'Holding' (4), 'Held' (5) and 'Restarting' (6) states into the return state register. This works with Cfg_SCTarget[6] = -1. When in 'Starting' or 'Running' the return state register is updated to note where it was. When a 'Hold' command is issued the state machine transitions through the holding chain of states but the return state register still has the 'Starting' or 'Running' state number in it. When 'Restarting' is done, it transitions to the state on the return state register.

Cfg_InhCmdOnIntlk = 2# 0000 0000 0000 0000 0000 0000 0000 0101

If the interlock is not OK, then inhibit the 'Start' (0) and 'Restart' (2) commands.

Generic Equipment Phase (raP_Opr_EPGen)

An equipment phase is a functional group of equipment that can conduct a finite number of specific minor processing activities when directed by a (recipe) phase.



For the object and visualization parameters, see PlantPAx Process Objects, publication [PROCES-RD200](#), and PlantPAx Visualization Files, publication [PROCES-RD201](#).

Guidelines

The raP_Opr_EPGen (Generic Equipment Phase Module) object controls an Equipment Phase in various command sources and monitors for fault conditions.

Use when:

- You want to group equipment, and you want to apply the ISA 88.01 state model using PhaseManager™
- You want to provide the following for a group of equipment
 - Apply a mode model to the equipment group
 - Apply interlocks and/or permissives to the group of equipment
 - Parameters that define the behavior of the group of equipment
 - Report / Resultant data from the group of equipment
 - A faceplate that allows monitoring / control of the equipment grouping
 - Monitor step (description), and allow forcing of steps in maintenance command source
 - Allow alarms to be defined for certain process / equipment failure conditions
 - Alarming function, including alarms based on device failure.

Functional Description

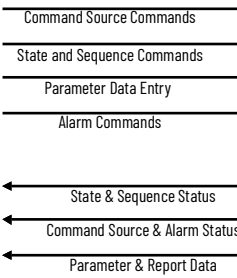
Phase Manager Program

Dispatch

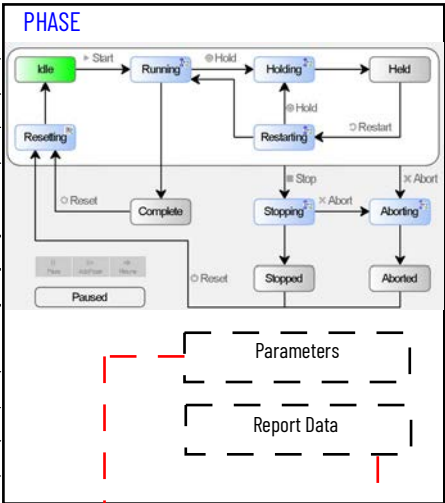
Contains raP_Opr_EPGen instruction and any external instructions required.



Operator

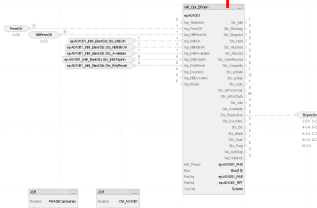


raP_Opr_EPGen



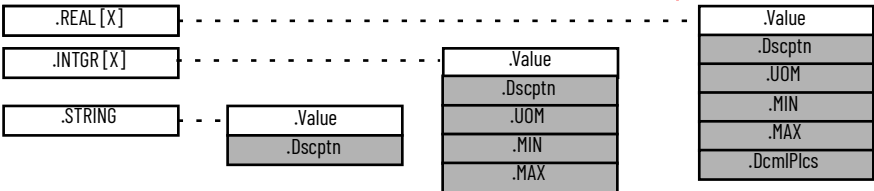
PHASECommands

Contains commands from your logic to raP_Opr_EPGen (as required)
Note: FactoryTalk® Batch issues commands directly to raP_Opr_EPGen via PhaseManager no logic is required.



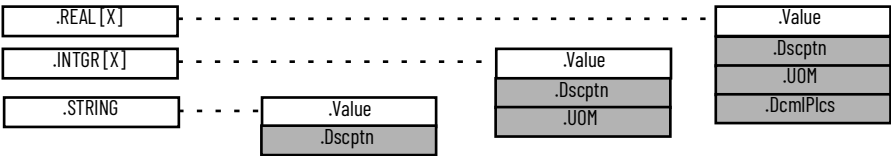
Parameters

Contains logic that maps parameters to raP_Opr_EPGen to Phase Manager tags (Input)



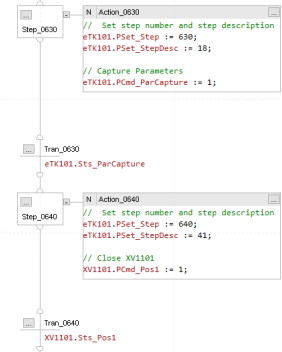
Reports

Contains logic that maps report data from raP_Opr_EPGen to Phase Manager tags (Output)



Phase State Routines

Contains your logic that sequences and coordinates devices (implement states as required)



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 raP_Opr_EPGen_5.30.**00**.A01.L5X Add-On Instruction must be imported into the controller project to be used in the controller configuration. The maintenance release number (boldfaced) can change as maintenance revisions are created.

Visualization Files

See [Library Versions on page 15](#) for general information on visualization files.

Operations

The primary operations of the raP_Opr_EPGen (Generic Equipment Phase Module) are to:

- Provides ISA 88 states, and associated commands
- Provides program structure as a container for coordination and sequencing logic.
- Provides Parameter display, and data entry (operator command source).
- Provides resultant (report) data display.
- Allow monitoring of process Step, and display Equipment Phase status.
- Monitor permissives, helping prevent Equipment Phase operation.
- Monitor interlock conditions to help prevent Equipment Phase operation or create failure condition.
- Provide a stepping mechanism, including the ability to force steps (maintenance)
- Monitor various Equipment Phase failure conditions, and produce alarms.
- Operate in maintenance, program, and operator command sources.
- Provide an “available” status for use by automation logic, to indicate that the Equipment Phase is available for operation.
- Provide a propagation mechanism to allow the Equipment Phase to publish status to and receive status from a group of equipment.
- Provides an interface to parameter display, data entry, and configuration.
- Provides an interface to resultant (report) data display and configuration.
- Provides interface to Prompt Response and configuration

Command Sources

The raP_Opr_EPGen (Generic Equipment Phase Module) uses the standard command sources that are implemented using an embedded PCMDSRC instruction. See PlantPAx Process Control Instructions, publication [PROCES-RM215](#) for more information.

PhaseManager

The raP_Opr_EPGen (Generic Equipment Phase Module) is designed to operate with PhaseManager™.

PhaseManager provides the following:

- ISA 88 state model, which can be executed by FactoryTalk® Batch, Studio 5000 Logix Designer®, or program logic.
- Program Structure, which contains phase state routines
- Program scoped tags, which allow individual tags to be configured as Input (parameters from FTBatch) or Output (resultant data, or report data, to FTBatch) for a particular PhaseManager phase.
- Phase data structure, which allows interface to the PhaseManager phase
- An instruction set for issuing commands, and controlling the execution of the PhaseManager phase.

The raP_Opr_EPGen (Generic Equipment Phase Module) provides an embedded reference to an associated PhaseManager Phase. The raP_Opr_EPGen (Generic Equipment Phase Module):

- Provides a human machine interface (faceplate), which allows control and monitoring of the PhaseManager phase.
- Provides a predefined human machine interface (faceplate), which allows input and monitoring of parameters (linked to program tags) and monitoring of resultant/report data (linked to program tags)
- Provides a normalized logic command interface, which utilizes the PhaseManager instruction set.

Program Structure

The raP_Opr_EPGen (Generic Equipment Phase Module) utilizes the PhaseManager program structure, as follows:

Routine	Description
Dispatch	Contains raP_Opr_EPGen instance, external function instances (Interlock, Permissive, Associated Device), and Phase State routinecalls. Phase state routines are a component of a PhaseManager Phase. <ul style="list-style-type: none"> • One or all available Phase state routines may be implemented. • The logic within a particular Phase state routine is executed when the Phase is in the corresponding state. • Any implemented Phase state routine requires a Phase State Complete instruction (which the state engine uses to determine the state is complete).
Aborting	Used for shutting down equipment in an emergency situation. If you have implemented Stopping, you would at a minimum duplicate the stopping logic within Aborting. In some cases, the sequence in an emergency situation (Aborting) differs from the orderly shutdown of equipment (Stopping).
Holding	Used if equipment or a subset of equipment must be shut down when the phase enters the hold state. It may also be advantageous to release owned equipment if maintaining ownership while held constrains production by maintaining ownership of shared equipment.
Resetting	Used to perform "clean-up" activities such as release owned equipment, etc.
Restarting	Implemented if Holding is implemented. Used to bring equipment from the state that it is in at the end of the Holding state back to the state that it was in prior Holding. Used in conjunction with PSet_HoldIdx and Val_LastRunningStep to return execution to the proper sequence step.
Running	Use to start up equipment, and acquire ownership of equipment (if necessary).
Stopping	Use if equipment must be shut down in a given sequence.
AlarmsSuppress	Contains EP_GEN alarm suppression logic.
Interlocks	Contains EP_GEN interlock mapping from interlock conditions to <EP_GEN>_Intlk block.
Parameter	Contains EP_GEN parameter mapping to and from Parameter blocks (_ParRpt (Enum, Integer, Real, String)) to EP_GEN instance.
Permissives	Contains EP_GEN permissive mapping from permissive conditions to <EP_GEN>_Perm block.
<EP_GEN>_PhaseCommands	Maps commands from EP_GEN to PhaseManager commands

Routine	Description
<EP_GEN>_PRXQ	PRXQ Routine container. Use the PRXQ instruction to initiate communication with FTBatch software.
Reports	Contains EP_GEN report mapping to and from Parameter blocks (_ParRpt (Enum, Integer, Real, String)) to EP_GEN instance.
ExtddAlarms	Contains EP_GEN instances of external alarm instances and trigger logic.

IMPORTANT Routines listed in the preceeding table are located within the PhaseManager program. This represents an example for implementing PhaseManager with the raP_Opr_EPGen (Generic Equipment Phase Module). PhaseManager may be implemented without the raP_Opr_EPGen (Generic Equipment Phase Module), in which case the PhaseManager program may be structured as desired. See the PhaseManager User Manual, Publication [LOGIX-UM001](#).

Alarms

The raP_Opr_EPGen Instruction uses the following alarms, which are implemented by using Tag Based Alarms.

Alarm	Alarm Name	Description
Device alarms	Alm_DvcAlms	Raised when a device within the Equipment Phase has an alarm.
Interlock trip	Alm_IntlkTrip	Raised when an interlock condition triggers a change in state of the Equipment Phase.
Report data	Alm_RptData	Raised when new report data are available.

Virtualization

The raP_Opr_EPGen Instruction has no Virtualization capability.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (False Rung)	Handle processing for EnableIn False (False Rung) the same as if the Equipment Module were Disabled by Command. The Equipment Module outputs are de-energized and the Equipment Module is shown as Disabled on the HMI.
Powerup (Pre-scan, First Scan)	Handles processing of command sources and alarms on Pre-scan and Powerup. On Powerup, the Equipment Module is treated as if it were Commanded to Reset all Program and Operator command.
Postscan (SFC Transition)	No SFC Postscan logic is provided.

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



ATTENTION: Disabling the raP_Opr_EPGen Add-On Instruction causes Equipment Phase outputs to become de-energized.

Local Message

The object raP_Opr_EPGen utilizes local message display elements to display Step Names, Material Names, and Summary information. A default local message file is provided for each information type. This default local message file populates the local message display elements from tags in the controller. For Step Names and Material names, these are the same controller tags that are used in previous versions of the library. The difference is that 512 messages are available, rather than the 99 messages in the previous version. To upgrade from previous versions, developers must add the local message file to the project and set the @Navigation property of the specified tag to the Local Message file name (see table below).

Information	Default Local Message File	File Name Reference	Default Controller Data
Material Name	SystemMaterialNames	Sts_eMtrl.@Navigation	System.Enum.Materials[x].@Description
Step Description	SystemStepDescriptions	Sts_eStep.@Navigation	System.Enum.Step_Desc[x].@Description
Summary Information	SystemSummary	Sts_eSummary.@Navigation	System.Enum.Summary_Desc[1].@Description

Users may add customized local messages for individual objects by creating a new local message file and populating the file with the customized strings or tag references. Then set the @Navigation property of the specified tag to the name of the new custom file.

FactoryTalk Optix Local Message

The object raP_Opr_Unit uses the @Label property of the specified tag to input the path for displaying Step Names, Material Names, and Summary information. FactoryTalk® Optix™ does not require Local Message files. For Step Names, Material Names, and Summary, these are the same controller tags that are used with FactoryTalk® View SE. FactoryTalk Optix directly retrieves Controller Data using the path that is specified in the @Label property of Logix Designer. Users must set the path information into @Label property of the specified tag to a Controller Data Path (see the following table).

Information	Reference	Customized Controller Data Path	Default Controller Data
Material Name	Sts_eMtrl@Label	System/Enum/Materials	System.Enum.Materials[x].@Description
Step Description	Sts_eStep@Label	System/Enum/Step_Desc	System.Enum.Step_Desc[x].@Description
Summary Information	Sts_eSummary@Label	System/Enum/Summary_Desc	System.Enum.Summary_Desc[1].@Description

Users may add Customized Controller Data for individual objects by creating a new tag member with the Data Type "raP_UDT_Opr_System" in Logix Designer.

Name	Alias For	Base Tag	Data Type	Description	External Access
System			raP_UDT_Opr_System	System Global Structure	Read/Write
System.Enum			raP_UDT_Opr_System_Enum	System Global Structure Enumerations	Read/Write
System.Enum.Step_Desc			BOOL[512]	System Global Structure Step Descriptions	Read/Write
System.Enum.Materials			BOOL[512]	System Global Structure Material Names	Read/Write
System.Enum.Summary_Desc			BOOL[512]	System Global Structure Summary Descriptions	Read/Write
System.Proj			raP_UDT_Opr_System_Proj	System Global Structure Project Settings	Read/Write
System.Sts			raP_UDT_Opr_System_Status	System Global Structure Status	Read/Write
NewCreatedSystem			raP_UDT_Opr_System	System Global Structure	Read/Write

Then set the @Label property of the specified tag to the Customized Controller Data Path and import the tag with the customized extended properties into FactoryTalk Optix.

Name	Value	Force	Style	Data Type	Description	Properties
MyUnit.XCmd_RptClear		0	Decimal	BOOL	Process U	Extended Properties
MyUnit.XCmd_RptRestore		0	Decimal	BOOL	Process U	Engineering Unit
MyUnit.Out_ExtddAmFeed		0	Decimal	BOOL	Process U	External Logging
MyUnit.Out_ExtddAmFeedAckAll		0	Decimal	BOOL	Process U	Label
MyUnit.Out_ExtddAmDisable		0	Decimal	BOOL	Process U	Navigation
MyUnit.Out_ExtddAmEnable		0	Decimal	BOOL	Process U	File
MyUnit.Out_ExtddAmSuppress		0	Decimal	BOOL	Process U	Required
MyUnit.Out_ExtddAmUnSuppress		0	Decimal	BOOL	Process U	Visible
MyUnit.Out_ExtddAmUnShelve		0	Decimal	BOOL	Process U	Alarms
MyUnit.Val_Actl	45.62		Float	REAL	Process U	Data
MyUnit.Sts_eSummary		4	Decimal	DINT	Process U	Value
MyUnit.Sts	240000_0000_0000_0000_...		Binary	DINT	Process U	External Logging
MyUnit.Sts_Alm		0	Decimal	BOOL	Process U	Label
MyUnit.Sts_AlmInb		0	Decimal	BOOL	Process U	Navigation
						Force Mask

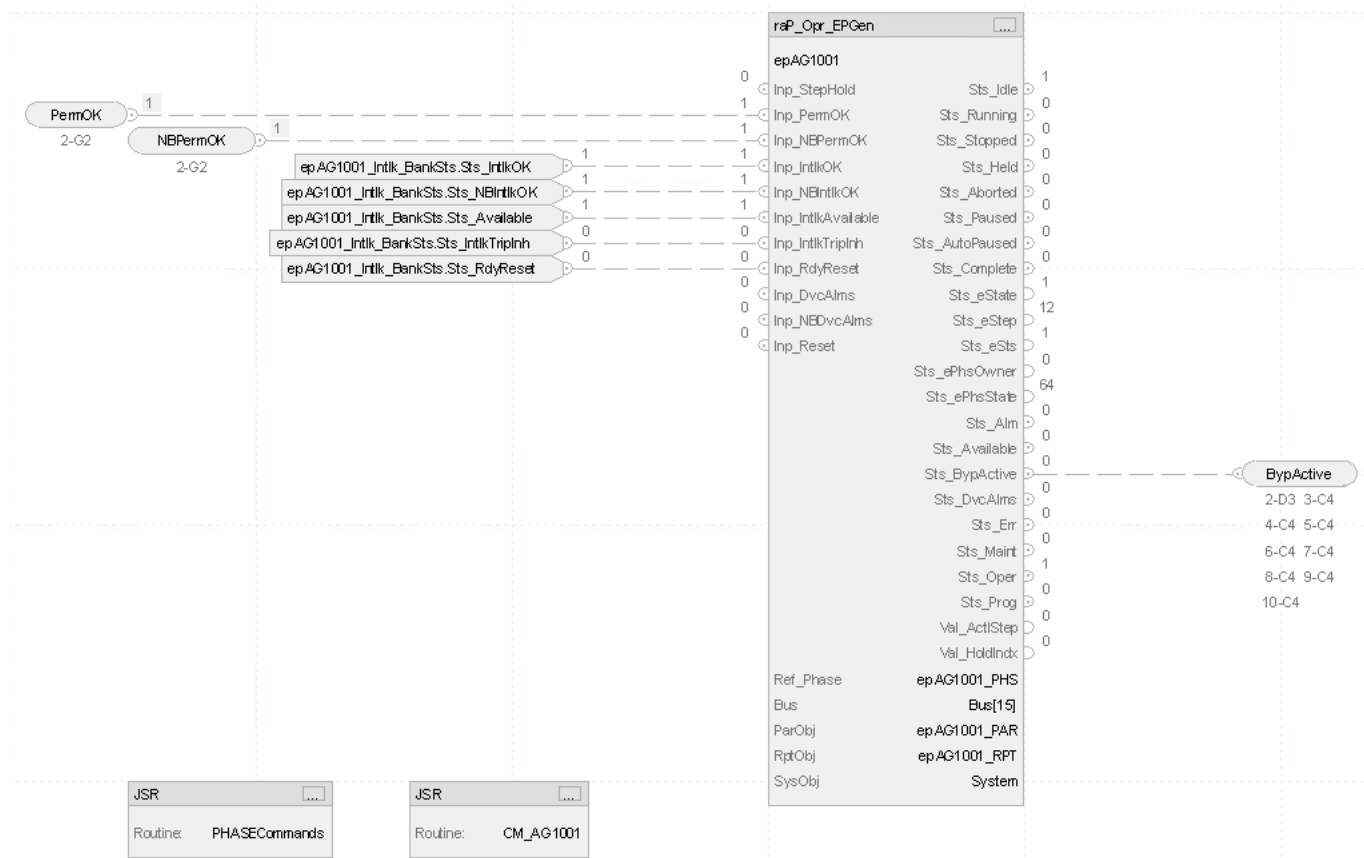
Tag Extended Properties and Default Alarm Settings

Common	
raP_Opr_EPGen.@Description	"Generic Equipment Phase"
raP_Opr_EPGen.@Area	"Area01"
raP_Opr_EPGen.@Instruction	"raP_Opr_EPGen"
raP_Opr_EPGen.@Label	"Equipment Phase"
raP_Opr_EPGen.@Library	"raP-5_30"
raP_Opr_EPGen.@URL	"n/a"
raP_Opr_EPGen.Cfg_HasMoreObj.@Navigation	" "
raP_Opr_EPGen.Sts_eStep.@Navigation	"SystemStepDescriptions"
raP_Opr_EPGen.Sts_eSummary.@Navigation	"SystemSummary"
raP_Opr_EPGen.Cfg_HasDvcAlmsObjNav.@Navigation	" "

General	
#2.Cfg_HasDetailDisplay.@Navigation	" "
#2.Sts_ExtddAlms.@Label	"Extended alarm"

Alarms		Alarm Default Message	Severity
raP_Opr_EPGen.Sts_DvcAlms.@Label	"Device alarm"	"/*S:0 %.@Description*/: Device alarm	500
raP_Opr_EPGen.Sts_IntlkTrip.@Label	"Interlock trip"	"/*S:0 %.@Description*/: Interlock trip	500
raP_Opr_EPGen.Sts_RptData.@Label	"Report data not collected"	"/*S:0 %.@Description*/: Report data	500

Programming Example



Notes:

Parameter and Reports (raP_Tec_ParRpt)

The raP_Tec_ParRpt Add-On Instruction is used to implement parameter data items. The raP_Tec_ParRpt instruction may be used as follows:

- For a read-only parameter
- For a read/write parameter
- For a parameter of type Integer, Real, String, or Enumeration
- Equipment Module (raP_Opr_EMGen) and Equipment Phase (raP_Opr_EPGen) are designed to work with the raP_Tec_ParRpt instruction



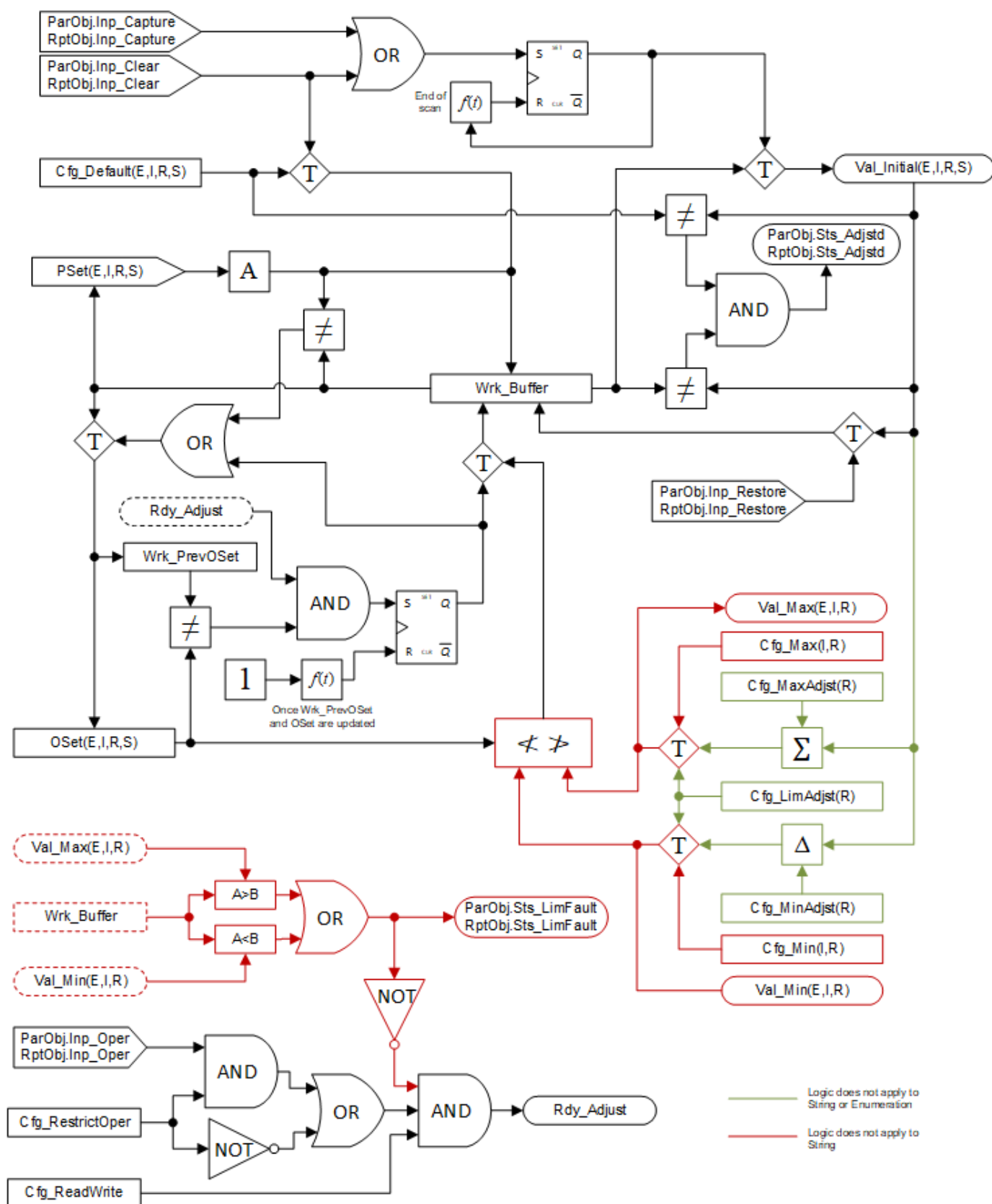
For the object and visualization parameters, see PlantPax Process Objects, publication [PROCES-RD200](#), and PlantPax Visualization Files, publication [PROCES-RD201](#).

Guidelines

Use when:

- You need the ability to view or modify a parameter from either the HMI or from logic
- You must arbitrate parameter input based on mode
- You need the ability to limit the value of a parameter, from either the HMI or logic
- You need the ability to capture an initial parameter value (based on a trigger), and provide an indication if the parameter was adjusted from the initial value
- You must limit the adjustment of a parameter within a deadband relative to an initial value
- You must apply command confirmation (that is, Electronic Signature) to parameter entry from the HMI.
- Your parameter is read-only or read/write
- You need a Parameter (recipe) or Report (resultant) parameter
- Your parameter is of data type: Integer, Real, String, or is an Enumeration

Functional Description



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 raP_Tec_ParRpt_5.30.**00**_A01.L5X or (raP_Tec_ParRpt_5.20.**00**_A01.L5X) Add-On Instruction must be imported into the controller project to be used in the controller configuration. The maintenance release number (boldfaced) can change as maintenance revisions are created.

Visualization Files

See [Library Versions on page 15](#) for general information on visualization files.

Operations

The primary operations of the raP_Tec_ParRpt (Parameter Instruction) are:

- Captures the initial value of the parameter (snap shot) when the Trigger goes TRUE. Maintains the initial value until the Clear input goes TRUE.
- Permits or denies Operator adjustment of the parameter value. When permitted, allows the adjustment of the parameter value within a deadband of the initial parameter value based on configured limits.
- Compares the initial parameter value to the present parameter value and produces an "Adjusted" status.
- Allows initial parameter values to be restored, when the Reset to Initial input goes TRUE.
- Limits the value of the parameter based on configured Minimum and Maximum limits, and produces a status when the parameter value is beyond limits.
- Allows the parameter to be configured as Read, or Read/Write
- Allows a default parameter value to be configured, restores defaults when Clear input goes TRUE.
- Allows the configuration of a text description, and units of measure (engineering units) for the parameter.
- When configured to allow Operator entry and read/write, and "Operator" mode input is true; produces "Ready to Adjust" status, and allows the parameter value to be entered from the HMI.
- Allows command confirmation to be applied to parameter entry from the HMI: No Signature, Performer Signature only, or Performer and Approvers Signatures.

Command Sources

The raP_Tec_ParRpt instruction does not have command sources. However, the raP_Tec_ParRpt provides an input to monitor for Operator mode, and uses this to arbitrate request to modify the parameter value.

Alarms

The raP_Tec_ParRpt instruction does not generate any alarms.

Virtualization

The raP_Tec_ParRpt Instruction has no Virtualization capability.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (False Rung)	Handle processing for EnableIn False (False Rung) the same as if the Equipment Module were Disabled by Command. The Equipment Module outputs are de-energized and the Equipment Module is shown as Disabled on the HMI.
Powerup (Pre-scan, First Scan)	Handles processing of modes and alarms on Pre-scan and Powerup. On Powerup, the Equipment Module is treated as if it were Commanded to Reset all Program and Operator command.
Postscan (SFC Transition)	No SFC Postscan logic is provided.

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



ATTENTION: Disabling the raP_Tec_ParRpt Add-On Instruction causes Equipment Phase outputs to become de-energized.

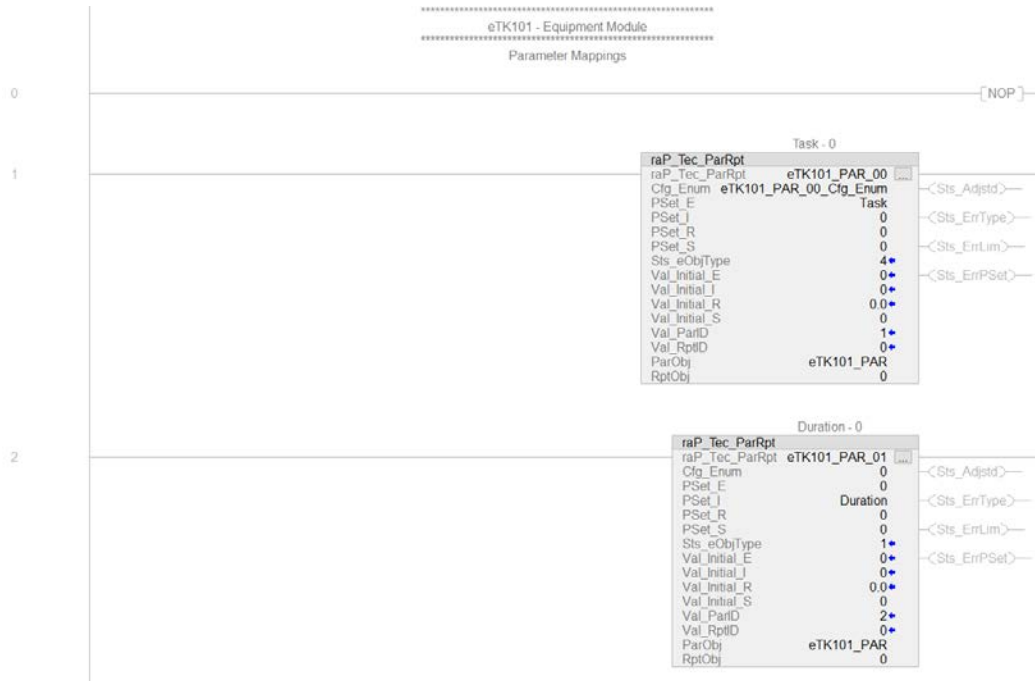
Tag Extended Properties and Default Alarm Settings

Common	
raP_Tec_ParRpt_PAR_XX.@Description	"Parameter"
raP_Tec_ParRpt_PAR_XX.@Area	"Area01"
raP_Tec_ParRpt_PAR_XX.@Instruction	"raP_Tec_ParRpt"
raP_Tec_ParRpt_PAR_XX.@Label	"Parameter Label"
raP_Tec_ParRpt_PAR_XX.@Library	"raP-5_30"
raP_Tec_ParRpt_PAR_XX.@URL	"n/a"
raP_Tec_ParRpt_PAR_XX.@EngineeringUnit	"%"
raP_Tec_ParRpt_RPT_XX.@Description	"Report"
raP_Tec_ParRpt_RPT_XX.@Area	"Area01"
raP_Tec_ParRpt_RPT_XX.@Instruction	"raP_Tec_ParRpt"
raP_Tec_ParRpt_RPT_XX.@Label	Report Label"
raP_Tec_ParRpt_RPT_XX.@Library	"raP-5_30"
raP_Tec_ParRpt_RPT_XX.@URL	"n/a"
raP_Tec_ParRpt_RPT_XX.@EngineeringUnit	"%"

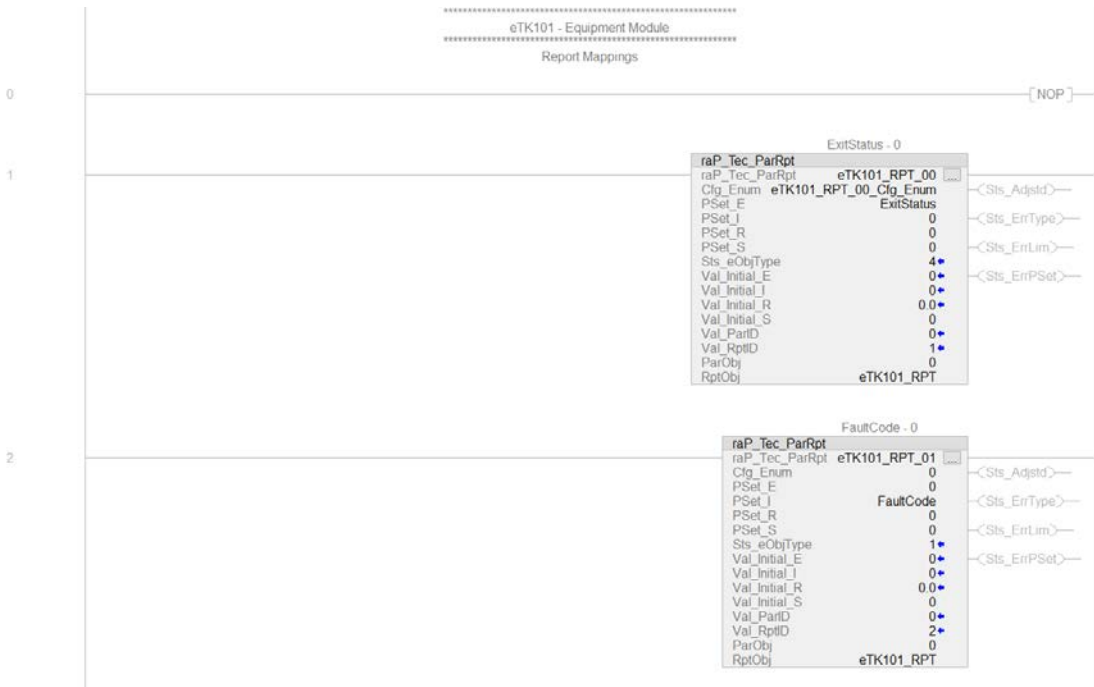
Programming Example

The example in the Function Description section shows the basic use of the raP_Opr_ParRpt Add-On Instruction. Typically, the raP_Opr_ParRpt instruction is not used on its own. The instruction is used with the Unit, EMGen, or the EPGen instructions, the instruction is used for both the Parameters and Reports. Multiples of each can be created as long as it follows the naming convention and has a unique number that is associated with it. The raP_Opr_ParRpt instruction allows for four different options, Enumeration, String, Integer, or Real, only one of these types can be used per instance. When used with the EPGen, the PSet parameter tag is associated to the FTBatch parameter or report.

Parameter Program Example



Reports Program Example



Operator Prompt (raP_Opr_Prompt)

The raP_Opr_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 Sequencer Add-On Instruction instruction also uses the prompt instruction. For more information on the Sequencer instruction, see Rockwell Automation Sequencer Object, Publication [PROCES-RM202](#).

Guidelines

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

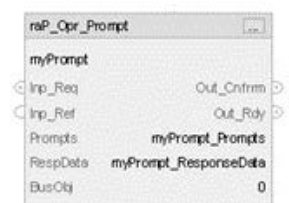
A prompt requires a response, but does not advise of an abnormal situation.

	Normal Operation	Abnormal Situation
Operator Response Not Required	Normal values and status	Alert
Operator Response Required	Prompt (raP_Opr_Prompt)	Alarm

Functional Description

The RespData tag at the bottom of the raP_Opr_Prompt function block lets you define where to store operator responses. This tag stores any operator response as a string in the application. This tag must be unique to every instance of the raP_Opr_Prompt instruction.

The optional BusObj tag allows the prompt to participate on the organizational bus. The entry should be a unique bus element in the bus array. With this field populated, an 'operator attention' indicator is propagated up through any organizational tree in which this bus element is assigned.



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 Files

The raP_Opr_Prompt_5.30.**00**_AOI.L5X or (raP_Opr_Prompt_5.20.**00**_AOI.L5X) Add-On Instruction must be imported into the controller project to be able to be used in the controller configuration. The maintenance release number (boldfaced) can change as maintenance revisions are created.

Visualization Files

See [Library Versions on page 15](#) for general information on visualization files.

Operations

Command Sources

The raP_Opr_Prompt Add-On Instruction does not use command sources.

Alarms

The raP_Opr_Prompt Instruction uses the following alarm, which is implemented by using Tag Based Alarms.

Alarm	Alarm Name	Description
Alert timeout	Alm_AlertTimeOut	Raised when no response to a posted prompt has been entered within the configured time.

Virtualization

The raP_Opr_Prompt Instruction has no Virtualization capability.

Tag Extended Properties and Default Alarm Settings

Common	
raP_Opr_Prompt.@Description	"Operator Prompt"
raP_Opr_Prompt.@Area	"Area01"
raP_Opr_Prompt.@Instruction	"raP_Opr_Prompt"
raP_Opr_Prompt.@Label	"Prompt"
raP_Opr_Prompt.@Library	"raP-5_30"
raP_Opr_Prompt.@URL	"n/a"
raP_Opr_Prompt.Cfg_HasMoreObj.@Navigation	" "

Alarms		Alarm Default Message	Severity
raP_Opr_Prompt.Sts.AlertTimeOut.@Label	"Prompt timed out"	/*S:0 %.@Description*/ Prompt time out alarm	500

raP_Opr_Prompt_Core

Common	
raP_Opr_Prompt.@Description	"Operator Prompt"
raP_Opr_Prompt.@Area	"Area01"
raP_Opr_Prompt.@Instruction	"raP_Opr_Prompt"
raP_Opr_Prompt.@Label	"Prompt"
raP_Opr_Prompt.@Library	"raP-5_30"
raP_Opr_Prompt.@URL	"n/a"
raP_Opr_Prompt.Cfg_HasMoreObj.@Navigation	" "
raP_Opr_Unit.Sts.eMtrl.@Navigation	"SystemMaterialNames"
raP_Opr_Unit.Sts.eSummary.@Navigation	"SystemSummary"

Notes:

Logix Diagnostic Objects



For the object and visualization parameters, see PlantPax Process Objects, publication [PROCES-RD200](#), and PlantPax Visualization Files, publication [PROCES-RD201](#).

Logix Change Detector (raP_Dvc_LgxChangeDet)

The raP_Dvc_LgxChangeDet (Logix Change Detector) Add-On Instruction monitors another Logix controller on the network and checks for changes that impact operation. Changes that can be monitored include downloads, online edits, I/O forcing, and controller mode changes.

No visualization elements are supplied with the raP_Dvc_LgxChangeDet instruction.

Guidelines

Use this instruction if you want to monitor a Logix controller for changes, to be sure that the correct application is being run for regulatory, quality, or security reasons.

Do not use this instruction in these situations:

- You have only one Logix controller. The raP_Dvc_LgxChangeDet instruction is intended to be run in a controller other than the one being monitored. Although the raP_Dvc_LgxChangeDet instruction can be configured to monitor the controller in which it is running, because it runs in controller logic, it cannot detect when the controller in which it is running is placed in Program mode.
- You have software, such as FactoryTalk® AssetCentre that monitors controllers on a secured network. This software provides much more extensive change tracking and auditing than the raP_Dvc_LgxChangeDet Add-On Instruction.

Functional Description

The raP_Dvc_LgxChangeDet instruction includes a source protected Add-On Instruction for use with Studio 5000 Logix Designer® software, version 33 or later, and Logix controllers. This instruction is intended to be used in one Logix controller to monitor another controller for changes.

Although this instruction must be executed in a Logix controller with firmware revision 33 or later, it can monitor controllers running firmware revision 12 or later.

The raP_Dvc_LgxChangeDet instruction monitors a Logix controller for the following types of changes:

- New entries being made in the change log, such as the following:
 - Modify, insert, or delete logic in Run or Program mode
 - Accept, assemble, or cancel edits
 - Enable, disable, or remove forces
 - Reconfigure a module

- Change an output list
- Send the 'Set Attribute' MSG or 'SSV' to a controller object class or instance
- Send the 'Set Attribute List' MSG to a controller object class or instance
- Send the 'Set Attribute All' MSG to a controller object class or instance
- Apply attributes to a controller object class or instance
- Create, delete, or reset a controller object instance
- Download of a different application
- Partially import into an application
- Download of an application without logic changes (but saved configuration data that has changed)
- Download of an application that contains offline edits
- Restore an application from an external drive source, such as a Secure Digital (SD) card

This instruction also reports the following:

- Controller/application 'check' value for change detection
- Date and time on the controller clock (YYYY-MM-DD hh:mm:ss)
- Day of the week based on the controller date
- Controller keyswitch position and mode
- Major and minor fault indications

The `raP_Dvc_LgxChangeDet` instruction is provided as a rung import for installation. Importing this rung into your ladder diagram routine:

- Imports the Add-On Instruction definition
- Create an instruction instance.
- Creates and completes all required tags and data structures for the instruction.

IMPORTANT Once the rung is imported, and before downloading and running the application, set the path in each of the referenced Message structures to point to the Logix controller to be monitored.

The interval at which this instruction checks for changes and updates its status is configurable, from 1...60 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 Files

The `raP_Dvc_LgxChangeDet_5.30.00.RUNG.L5X` or (`raP_Dvc_LgxChangeDet_5.20.00.RUNG.L5X`) rung import file must be imported into the controller project for the controller that is performing the monitoring. It is not necessary to add any logic to the controller being monitored. The maintenance release number (boldfaced) can change as maintenance revisions are created.

Visualization Files

There are no visualization files because the `raP_Dvc_LgxChangeDet` object does not use Graphic Symbols or Faceplates.

Operations

Command Sources

The raP_Dvc_LgxChangeDet instruction has no commands or outputs that are intended to control equipment and therefore does not have any command sources.

Alarms

The raP_Dvc_LgxChangeDet Instruction uses the following alarm, which is implemented by using Tag Based Alarms.

Alarm	Alarm Name	Description
Change detected	Alm_ChangeDetected	Raised when a change in controller operation such as a download or online edit has been detected.

Virtualization

The raP_Dvc_LgxChangeDet Add-On Instruction does not have a Virtualization capability.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	No EnableIn False logic is provided. The raP_Dvc_LgxChangeDet instruction must always be scanned true. In relay ladder logic, the raP_Dvc_LgxChangeDet instruction must be by itself on an unconditional rung. If the Rung Import provided with the Rockwell Automation® Library is used to install this instruction, the proper rung is created for you.
Powerup (prescan, first scan)	On Prescan, any commands that are received before First Scan are discarded. The update timer and internal polling status are reset. On first scan, the Change Detected internal status latch is cleared.
Postscan (SFC transition)	No SFC Postscan logic is provided.

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

Tag Extended Properties and Default Alarm Settings

Common	
raP_Dvc_LgxChangeDet.@Description	"Logix Change Detector"
raP_Dvc_LgxChangeDet.@Area	"Area01"
raP_Dvc_LgxChangeDet.@Instruction	"raP_Dvc_LgxChangeDet"
raP_Dvc_LgxChangeDet.@Label	"Controller Name"
raP_Dvc_LgxChangeDet.@Library	"raP-5_30"
raP_Dvc_LgxChangeDet.@URL	"n/a"
raP_Dvc_LgxChangeDet.Cfg_HasMoreObj.@Navigation	" "

Alarms		Alarm Default Message	Severity
raP_Dvc_LgxChangeDet.Sts_ChangeDetected.@Label	"Logic change detected"	/*S:0 %.@Description*/: Controller logic change detected	1000

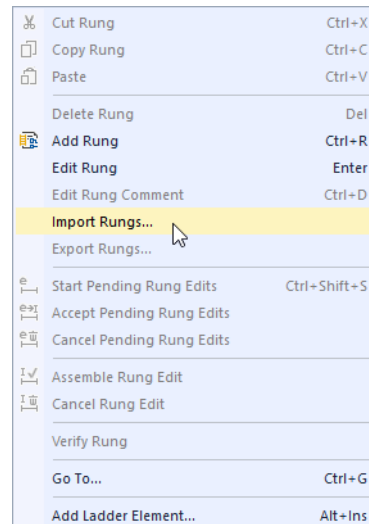
Programming Example

The `raP_Dvc_LgxChangeDet` instruction is provided fully configured as a rung import; so little programming is required for the instruction to be used. This programming example shows how the rung import is used to instantiate the `raP_Dvc_LgxChangeDet` instruction.

Since the `raP_Dvc_LgxChangeDet` instruction is a rung import, it must be created in a Ladder Diagram routine. By default, `raP_Dvc_LgxChangeDet` checks controllers for changes only every 5 seconds, so the ladder routine does not need to run in a fast periodic task.

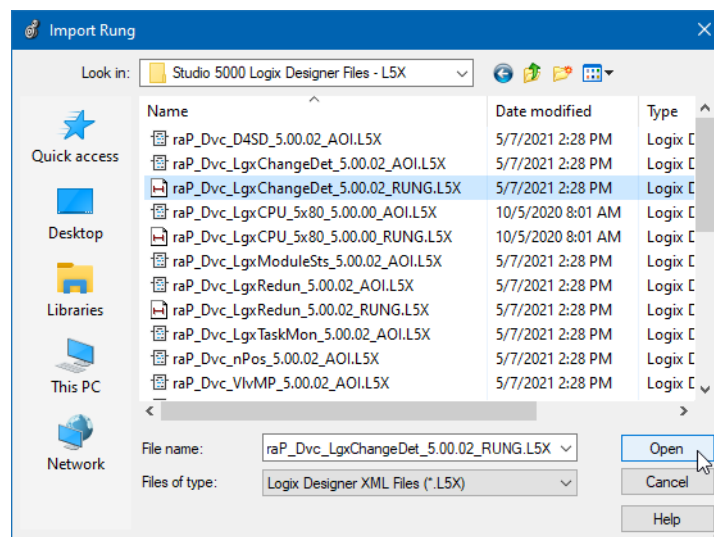
The following steps describe how you instantiate `raP_Dvc_LgxChangeDet` in your routine.

1. In your ladder routine, right-click where to insert the rungs and select Import Rungs.

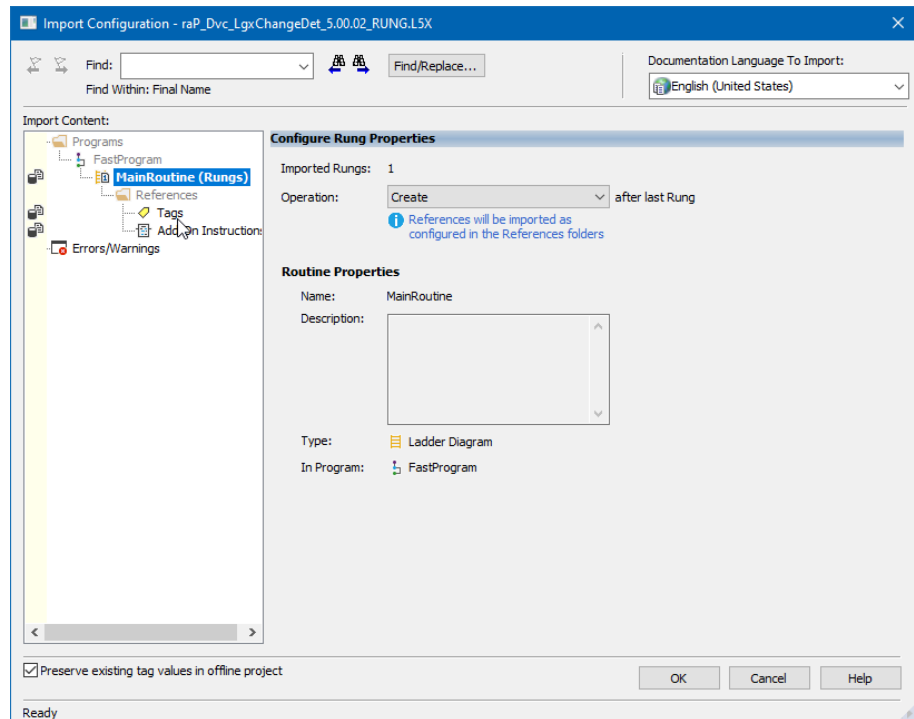


The Import Rungs dialog box appears.

2. Select the `raP_Dvc_LgxChangeDet` rung import file that is named in [Required Files on page 146](#).
3. Select Open.



The Import Configuration dialog box appears.

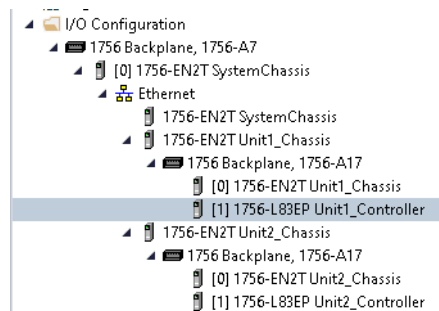


4. Rename the tags being imported to incorporate the name of the controller being monitored.

One controller can monitor several others. Adding the controller name to the tag makes it easier to track the individual instances when monitoring multiple controllers.

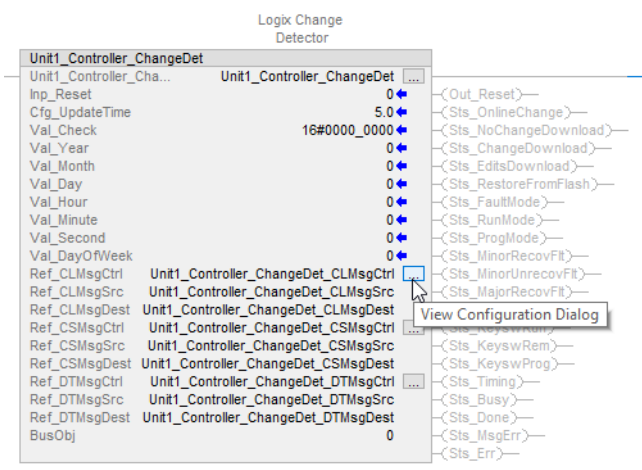
5. Select OK.
6. To point to the controller being monitored for changes, change the path in each of the MSG control tags.

If you create a link to the controller in the I/O tree configuration, enter the name that is assigned to that controller.



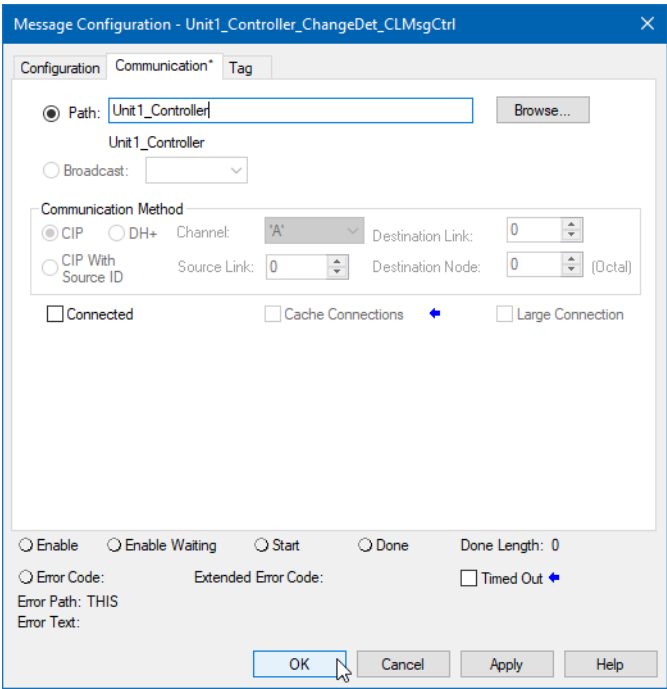
7. Complete the following steps for each of the three MSG control tags.

a. Select the ellipsis next to the MSG control tag.



The Message Configuration dialog box appears.

b. Select the Communication tab and change the path to the controller link created in the I/O tree.



c. Select OK.

8. Place the controller in RUN mode.

Status bits on the raP_Dvc_LgxChangeDet instruction indicate changes that are made to the monitored controller. Set Cmd_AckAll to 1 to clear the latched-in detections.

Logix Controller CPU Utilization (raP_Dvc_LgxCPU_5x80)

The raP_Dvc_LgxCPU_5x80 (Logix Controller CPU Utilization) Add-On Instruction monitors a Logix controller, and provides information on controller CPU utilization, communication usage, and other information. Data that is provided by the raP_Dvc_LgxCPU_5x80 instruction is useful to diagnose communication or control responsiveness issues and in tuning the performance of control tasks for optimum controller performance.

The raP_Dvc_LgxCPU_5x80 instruction can be loaded as part of a control application and disabled (default) until needed. The instruction can also be enabled at a slow update rate for general controller monitoring. The update rate can be increased, if necessary, as directed by a

Rockwell Automation Technical Support representative to help diagnose controller performance issues.

The global object and faceplate in the following image are examples of the HMI that is provided with this library object.

Guidelines

Use this instruction in these situations:

- Monitor general controller resource utilization:
 - Processor utilization
 - Communication capacity
 - Networking performance and connection usage
- Gather data to help resolve a specific issue under the direction of a Rockwell Automation Technical Support representative
- Tune the periods or priorities of multiple tasks in a controller to optimize control and observe how changes in task configuration affect CPU and other resource usage in the controller
- For use with CompactLogix® 5380 Controllers and ControlLogix 5580 Controllers. Firmware / software must be version 33 and later.

Do not use this instruction at a high update rate on a continuing basis. The raP_Dvc_LgxCPU_5x80 instruction increases the communication load on the controller when it is polling for performance data. At high update rates, the resource load that the raP_Dvc_LgxCPU_5x80 instruction polling generates can affect control performance, especially if you already have a fully loaded controller.

Functional Description

The raP_Dvc_LgxCPU_5x80 instruction collects and summarizes various data from the Logix controller that is being monitored. This information includes the following:

- Processor Identity information:
 - Catalog number and description
 - Major and minor firmware revision numbers
- Communication Responsiveness information:
 - CPU% used for responding to communication requests
 - Optimized Packets that are used for responding to communication requests

IMPORTANT	The raP_Dvc_LgxCPU_5x80 instruction does not support SoftLogix™ 5800 or RSLogix™ Emulate 5000 controllers.
------------------	--

- CPU utilization (%):
 - Continuous task (or unused CPU, if no continuous task)
 - Periodic and Event tasks
 - Responding to communication requests (such as from HMI)
 - System (I/O scan, timer updates, everything else)
- Communication connection usage:
 - Total connections available
 - Connections that are used for each of several classes of communication
 - Unconnected buffers and cached messages
- I/O Forcing status
- Controller minor faults

The items that are listed previously are displayed on several faceplate tabs, with summary information on the main (home) tab.

IMPORTANT

We recommend that you access the raP_Dvc_LgxCPU_5x80 faceplate when you contact Rockwell Automation Technical Support. The information on the Operator (home) tab is often requested when you call. You also need your Studio 5000 Logix Designer software serial number or other license or support contract information. The Maintenance tab has a space for you to record this information for reference.

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 Files

For use with Logix 5x80 controllers, one instance of raP_Dvc_LgxCPU_5x80_5.30.**00**_RUNG.L5X or (raP_Dvc_LgxCPU_5x80_5.20.**00**_RUNG.L5X) must be imported into the controller. It is recommended to only use the rung import when configuring the Add-On Instruction in a project. The maintenance release number (boldfaced) can change as maintenance revisions are created.

Visualization Files

See [Library Versions on page 15](#) for general information on visualization files.



For the object and visualization parameters, see PlantPAx Process Objects, publication [PROCES-RD200](#), and PlantPAx Visualization Files, publication [PROCES-RD201](#).

Operations

Command Sources

The raP_Dvc_LgxCPU_5x80 instruction is not intended to control equipment and therefore does not have any command sources. However, there are two program commands and two maintenance commands available to enable and disable collection of data (PCmd_Enable, PCmd_Disable, MCmd_Enable, MCmd_Disable). The maintenance commands are only available via the HMI faceplate.

Alarms

The raP_Dvc_LgxCPU_5x80 Add-On Instruction does not provide any alarms. If an alarm is required, define the output status to be alarmed as a Logix Tag Based Alarm.

Virtualization

The raP_Dvc_LgxCPU_5x80 Add-On Instruction does not have a Virtualization capability.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	The raP_Dvc_LgxCPU_5x80 instruction has no EnableInFalse logic and does nothing on a false rung. Data that are associated with the instruction are left in their last state.
Powerup (prescan, first scan)	Logic is sure that the window time is sent to the controller when it transitions to Run mode. Previously active polling (before power down or transition to Program mode) is canceled. High-water data that is stored in the instruction (not built in to the controller status registers) are cleared.
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.

Tag Extended Properties

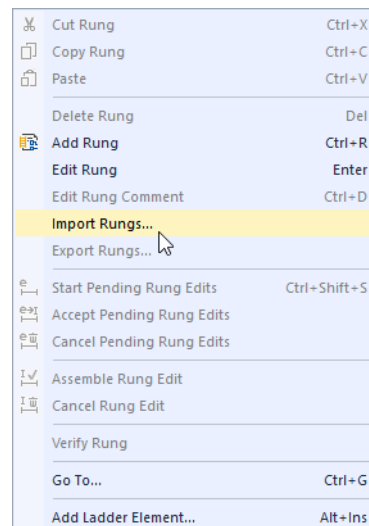
Common	
raP_Dvc_LgxCPU_5x80.@Description	"Processor utilization (5380/5580, version 33 and later)"
raP_Dvc_LgxCPU_5x80.@Area	"Area01"
raP_Dvc_LgxCPU_5x80.@Instruction	"raP_Dvc_LgxCPU_5x80"
raP_Dvc_LgxCPU_5x80.@Label	" "
raP_Dvc_LgxCPU_5x80.@Library	"raP-5_30"
raP_Dvc_LgxCPU_5x80.@URL	"n/a"
raP_Dvc_LgxCPU_5x80.Cfg_HasMoreObj.@Navigation	" "

Programming Example

The raP_Dvc_LgxCPU_5x80 instruction is provided fully configured as a rung import; therefore, little programming is required for the instruction to be used. This programming example shows how the rung import is used to instantiate the raP_Dvc_LgxCPU_5x80 instruction.

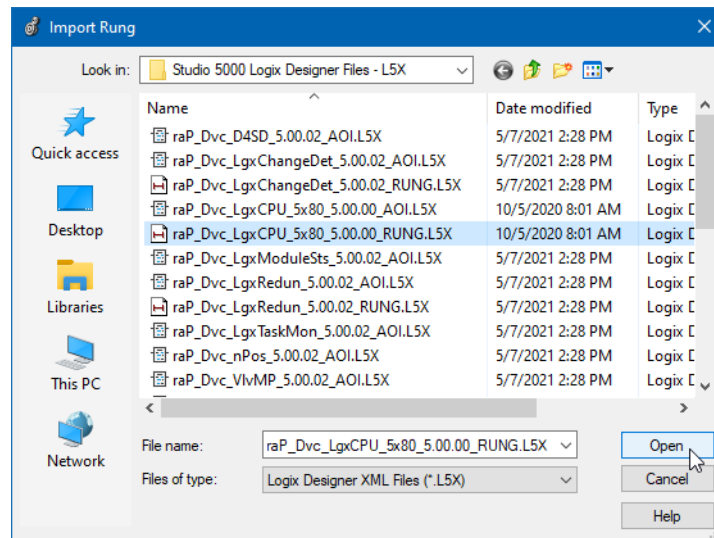
Because raP_Dvc_LgxCPU_5x80 is a rung import, it must be created in a ladder diagram routine. The following steps describe how to instantiate raP_Dvc_LgxCPU_5x80 in your routine.

1. In your ladder routine, right-click where to insert the rungs and select Import Rungs.



The Import Rungs dialog box appears.

2. Select the appropriate raP_Dvc_LgxCPU_5x80 rung import file that is named in [Required Files on page 152](#).

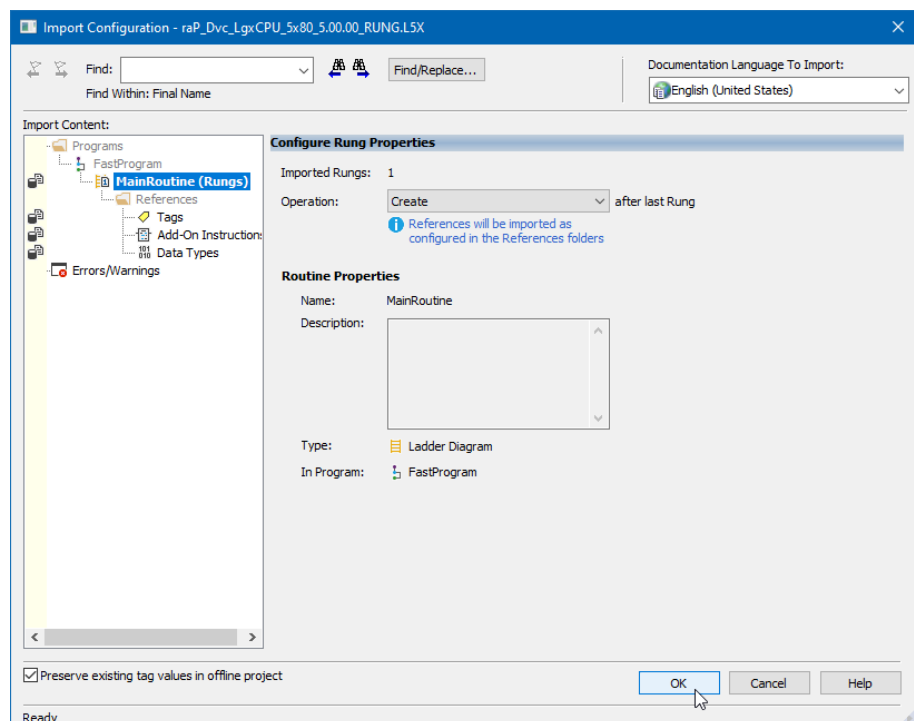


3. Select Open.

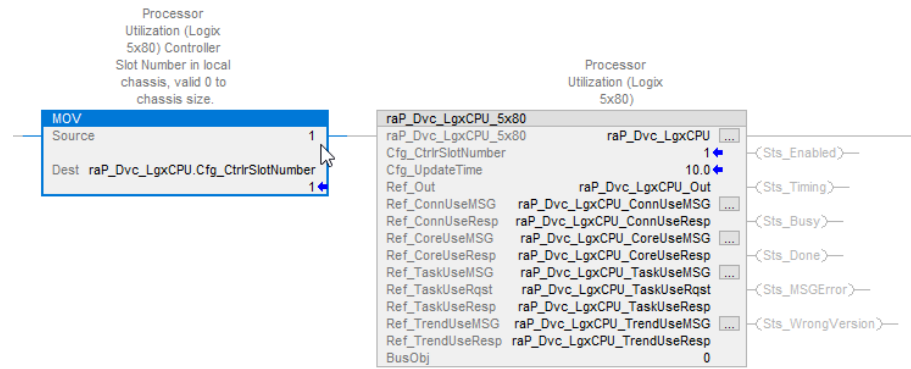
The Import Configuration dialog box appears.

IMPORTANT Do not change tag names in the Import Configuration. There must be one instance only of the raP_Dvc_LgxCPU_5x80 instruction in any controller project.

4. To create the instance of raP_Dvc_LgxCPU_5x80, select OK.



- Set the controller slot number in the Source of the MOV.



Set this value before putting the controller into Run mode. If the value is changed, it requires a transition from Program to Run on the controller for the new value to take effect.

6. Select the Finalize All Edits in Program icon.
7. To finalize all edits, Select Yes.

Logix Redundant Controller Monitor (raP_Dvc_LgxRedun)

The raP_Dvc_LgxRedun (Logix Redundant Controller Monitor) Add-On Instruction monitors one redundant pair of Logix controllers. The instruction checks primary and secondary controller status that can affect the ability of the system to switch to the back-up controller on a failure of the primary.

Guidelines

Use this instruction in these situations:

- You are using Logix controllers in a redundant configuration.
- You want to monitor the status of the redundant controller pair.
- You want to display this status to operators, maintenance personnel, or engineers.

Do not use this instruction in these situations:

- You are using single Logix controllers, not in a redundant configuration. The raP_Dvc_LgxRedun instruction is designed around the ControlLogix Enhanced Redundancy System architecture, by using information from the 1756-RM2 Redundancy Modules. The raP_Dvc_LgxRedun Add-On Instruction does not verify in a non-redundant system because the data items it monitors do not exist in a non-redundant configuration.
- Your controllers are in an accessible location and the indicators on the controllers, network modules, and redundancy modules provide sufficient information about redundancy status.

For more information, see the ControlLogix Enhanced Redundancy System User Manual, publication [1756-UM535](#).

Functional Description

The raP_Dvc_LgxRedun instruction is provided as a rung import for installation. Importing this rung into your ladder diagram routine:

- imports the Add-On Instruction definition
- creates an instruction instance
- creates and completes all required tags and data structures for the instruction

Once the rung is imported, and before you download and run the application, set the path in each Message tag that references the input/output parameters of the instruction to point to slot that contains the 1756-RM2 module in the local chassis ('1, <slot>').

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 Files

The raP_Dvc_LgxRedun_5.30.**00**_RUNG.L5X or (raP_Dvc_LgxRedun_5.20.**00**_RUNG.L5X) rung import file must be imported into the controller project to be able to be used in the controller configuration. The maintenance release number (boldfaced) can change as maintenance revisions are created.

Visualization Files

See [Library Versions on page 15](#) for general information on visualization files.



For the object and visualization parameters, see PlantPAx Process Objects, publication [PROCES-RD200](#), and PlantPAx Visualization Files, publication [PROCES-RD201](#).

Operations

The raP_Dvc_LgxRedun instruction monitors a redundant pair of Logix controllers and provides the following information and capabilities:

- Determines and displays whether the current primary controller is in Chassis 'A' or Chassis 'B' (as defined by user configuration)
- Displays the Chassis A and Chassis B Redundancy Module (1756-RM2) status
- Displays the Controller A and Controller B redundancy status
- Displays the Controller A and Controller B keyswitch positions
- Displays the overall compatibility between modules in Chassis A and modules in Chassis B
- Displays the synchronization progress in percent complete
- Displays the amount of data that is transferred from the Primary redundancy module to the Secondary in the most recent transfer, and the most sent in any transfer (high-water mark)

This instruction also supports the following commands, if enabled in the configuration:

- Initiate a switchover from Primary to Secondary
- Initiate a resynchronization of the system (if it does not take place automatically)

Command Sources

The raP_Dvc_LgxRedun instruction has no commands or outputs that are intended to control equipment and so does not have any command sources.

Alarms

The raP_Dvc_LgxRedun Instruction uses the following alarm, which is implemented by using Tag Based Alarms.

Alarm	Alarm Name	Description
Secondary not ready	Alm_SecNotRdy	Raised when the secondary controller in a redundant pair is not ready to take over control on loss of the primary.

Virtualization

The raP_Dvc_LgxRedun Add-On Instruction does not have a Virtualization capability.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	No EnableIn False logic is provided. The raP_Dvc_LgxRedun instruction must always be scanned true. In relay ladder logic, the raP_Dvc_LgxRedun instruction must be by itself on an unconditional rung. If the Rung Import provided with the Rockwell Automation is used to install this instruction, the proper rung is created for you.
Powerup (prescan, first scan)	On Pre-scan, any commands that are received before first scan are discarded.
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.

Tag Extended Properties and Default Alarm Settings

Common	
raP_Dvc_LgxRedun.@Description	"Logix Redundant Controller Monitor"
raP_Dvc_LgxRedun.@Area	"Area01"
raP_Dvc_LgxRedun.@Instruction	"raP_Dvc_LgxRedun"
raP_Dvc_LgxRedun.@Label	"Redundant Controller"
raP_Dvc_LgxRedun.@Library	"raP-5_30"
raP_Dvc_LgxRedun.@URL	"n/a"

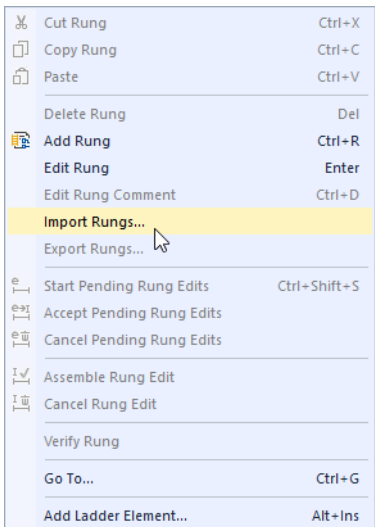
Alarms		Alarm Default Message	Severity
raP_Dvc_LgxRedun.Sts_SecNotRdy.@Label	"Secondary not ready"	Redundancy: secondary controller not ready to take control.	500

Programming Example

The raP_Dvc_LgxRedun instruction is provided fully configured as a rung import, so little programming is required for the instruction to be used. This programming example shows how the rung import is used to instantiate the raP_Dvc_LgxRedun instruction.

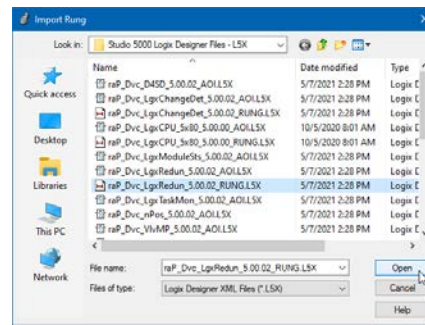
As raP_Dvc_LgxRedun is a rung import, it must be created in a Ladder Diagram routine. The following steps describe how you instantiate raP_Dvc_LgxRedun in your routine.

1. In your ladder routine, right-click where to insert the rungs and select Import Rungs.



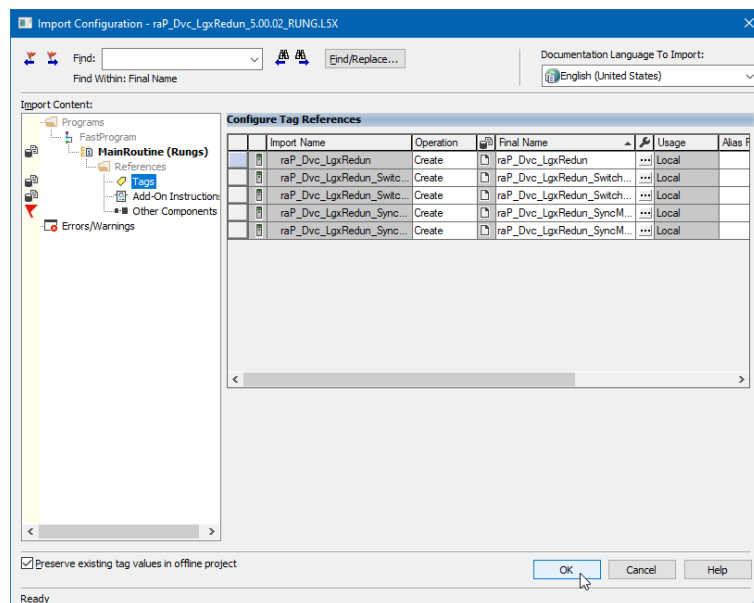
The Import Rungs dialog box appears.

2. Select the appropriate raP_Dvc_LgxRedun rung import file that is named in [Required Files on page 156](#).
3. Select Open.

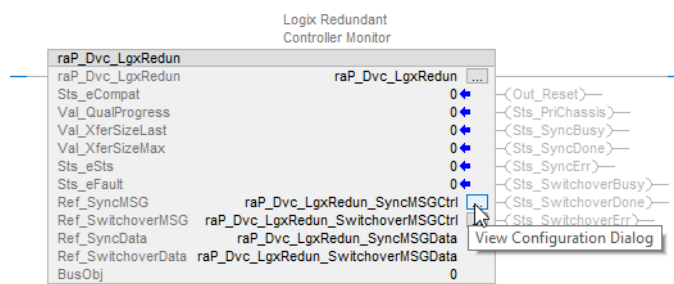


The Import Configuration dialog box appears.

4. To create the instance of raP_Dvc_LgxRedun, select OK.

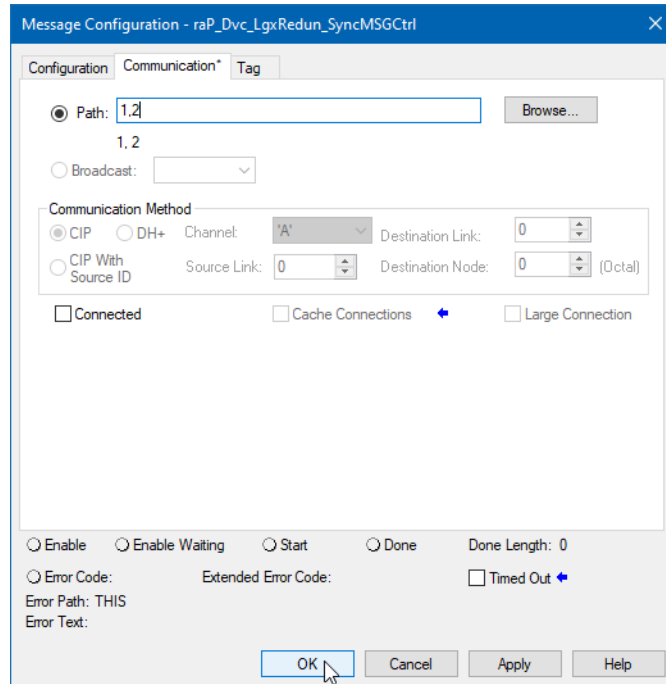


5. Complete the following steps for each of the two MSG controls to set the path to point to the 1756-RM2 module in the local chassis.
 - a. Select the ellipsis next to the MSG control tag.



The Message Configuration dialog box appears.

- b. To set the second number in the path to the slot number of the 1756-RM2 module, Select the Communication tab.



- c. Select OK.

Logix Module Status (raP_Dvc_LgxModuleSts)

The raP_Dvc_LgxModuleSts (Logix Module Status) Add-On Instruction monitors the connection status of one module or device in the I/O configuration tree of the Logix controller, and monitors it for any I/O channel faults on the module. The instruction provides an "I/O fault" status to dependent equipment, and provides a "Module Fault" status and alarm if the connection to the module is lost. It also provides an "Any Channel Fault" status and alarm if any I/O channel on the module reports a fault.

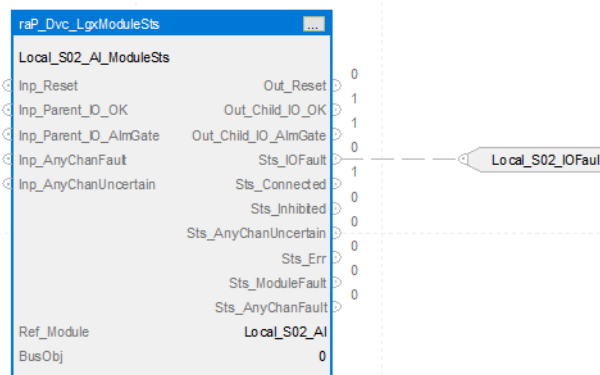
Guidelines

Use this instruction if you want to monitor the I/O connection status of a given module or device. This instruction is for use in CompactLogix 5380 and ControlLogix 5580 controllers using software / firmware revision 33 or later.

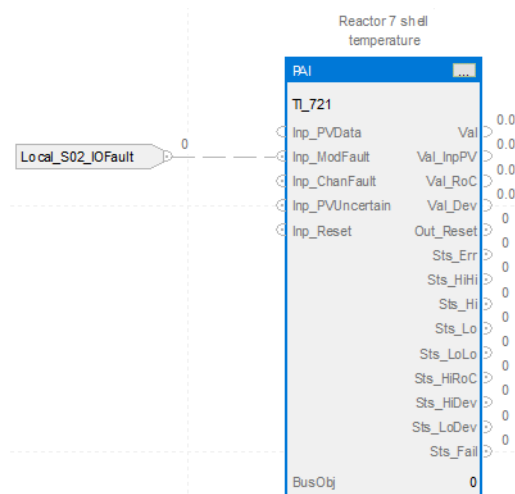
Functional Description

The raP_Dvc_LgxModuleSts Add-On Instruction is used to check the I/O connection status for the given module or device. The instruction provides an I/O Fault status output, which is 1 when the connection is lost, and 0 when the connection to the I/O module is OK and running normally. This status is available for use by other Add-On Instructions that use inputs or outputs of the given I/O module or device.

The following images show how the I/O Fault status output is connected to instructions that use the module being monitored. Here is the code showing the raP_Dvc_LgxModuleSts instruction getting the connection status for the module in Local chassis, Slot 2:



That status is passed along to the Analog Input instruction, which uses an input on that module:



The `raP_Dvc_LgxModuleSts` instruction can be used to provide the connection status for any connected device (anything with a Requested Packet Interval) in the I/O Configuration tree in Studio 5000 Logix Designer application. These devices include I/O modules, network communication modules, motor drives, overload relays, flowmeters, analyzers, weigh scales, and other devices on EtherNet/IP™ or another I/O network, or in the chassis that contains the controller.

IMPORTANT

Entry of a name for an I/O module or other device in the I/O Configuration is optional. However, in order for the `raP_Dvc_LgxModuleSts` instruction to refer to the module or device, you **MUST** give the module or device a name.

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 Files

The `raP_Dvc_LgxModuleSts_5.30.00_AOI.L5X` or (`raP_Dvc_LgxModuleSts_5.20.00_AOI.L5X`) Add-On Instruction definition file must be imported into the controller project to be able to be used in the controller configuration. The maintenance release number (boldfaced) can change as maintenance revisions are created.

Visualization Files

See [Library Versions on page 15](#) for general information on visualization files.

Operations

Command Sources

The `raP_Dvc_LgxModuleSts` instruction has no commands or outputs that are intended to control equipment and therefore does not have any selection of active command source.

Alarms

The `raP_Dvc_LgxModuleSts` Instruction uses the following alarms, which are implemented using Tag Based Alarms:

Alarm	Alarm Name	Description
Module Fault	Alm_ModuleFault	Raised when the I/O connection to the module or device is not in the Running state.
Any Channel Faulted	Alm_AnyChanFault	Raised when any I/O channel on the module is reporting a fault.

Virtualization

Virtualization allows the `raP_Dvc_LgxModuleSts` instruction to report a virtual connection status for use in test, demonstration, or training systems. The `raP_Dvc_LgxModuleSts` Add-On Instruction can be selected to virtual or physical (normal) operation. When physical operation is selected, the actual module connection status is monitored, and an I/O Fault status and Module Fault alarm is reported if the connection is not running. When virtual operation is

selected, the actual module connection status is ignored; the Set_VirtualConnectedSts input parameter determines the reported connection status.

Set_VirtualConnectedSts value	Description
1	Connected, the connection status is reported as OK
0	Faulted, the connection status is reported as faulted, the Sts_IOfault status is raised for dependent devices, and the Alm_ModuleFault alarm is raised.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	No EnableIn False logic is provided. The raP_Dvc_LgxModuleSts instruction must always be scanned true. In relay ladder logic, the raP_Dvc_LgxModuleSts instruction must be by itself on an unconditional rung.
Powerup (prescan, first scan)	All commands, including alarm acknowledge and reset, virtual or physical selection, maintenance bypass and check, plus all latched internal fault status bits, are cleared on prescan.
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.

Tag Extended Properties and Default Alarm Settings

Common	
raP_Dvc_LgxModuleSts.@Description	"Logix - Module Status"
raP_Dvc_LgxModuleSts.@Area	"Area01"
raP_Dvc_LgxModuleSts.@Instruction	"raP_Dvc_LgxModuleSts"
raP_Dvc_LgxModuleSts.@Label	"Module Name"
raP_Dvc_LgxModuleSts.@Library	"raP-5.30"
raP_Dvc_LgxModuleSts.@URL	"n/a"

Alarms		Alarm Default Message	Severity
raP_Dvc_LgxModuleSts.Sts_AnyChanFault.@Label	"I/O channel fault"	/*S:0 %.@Description*/: An I/O channel is faulted	500
raP_Dvc_LgxModuleSts.Sts_ModuleFault.@Label	"I/O module fault"	/*S:0 %.@Description*/: I/O module or device connection fault	500

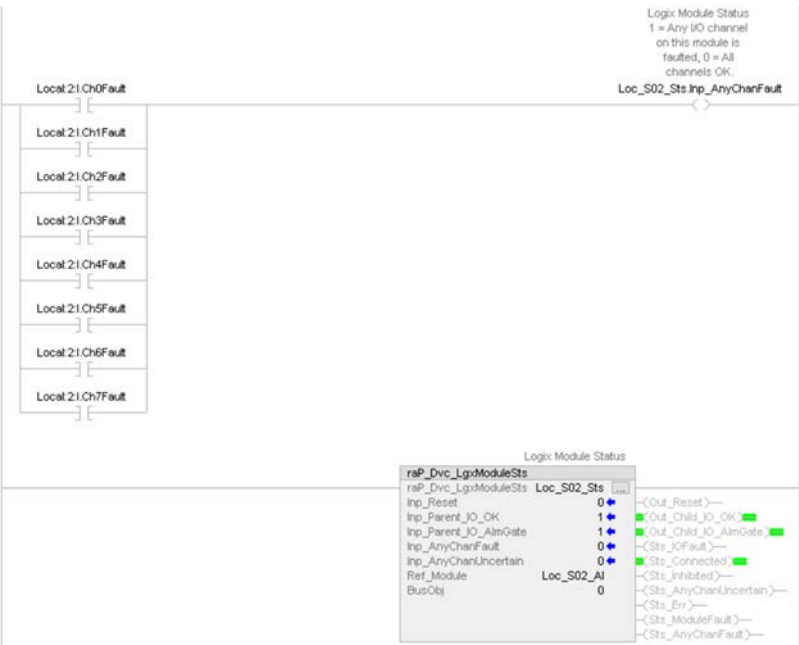
Programming Examples

The example in the Function Description section shows the basic use of the raP_Dvc_LgxModuleSts Add-On Instruction for monitoring a module connection. The instruction can also monitor and alarm channel faults on the I/O module, using some simple external logic. For many discrete modules, the individual channel fault bits are collected into a single INT or DINT (16-bit or 32-bit integer), and if the value of this integer is not zero, there is at least one channel fault:



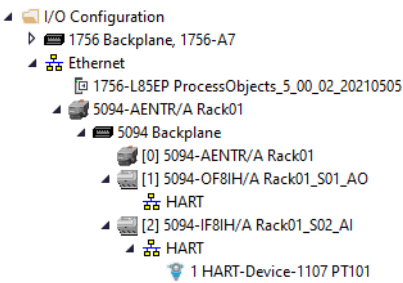
The "NEQ" instruction determines that there is at least one channel fault, and this status is forwarded to the raP_Dvc_LgxModuleSts instruction via the Inp_AnyChanFault input pin.

Some analog modules use a similar grouping of channel faults; others require the user to "OR" the individual channel faults in the external logic:



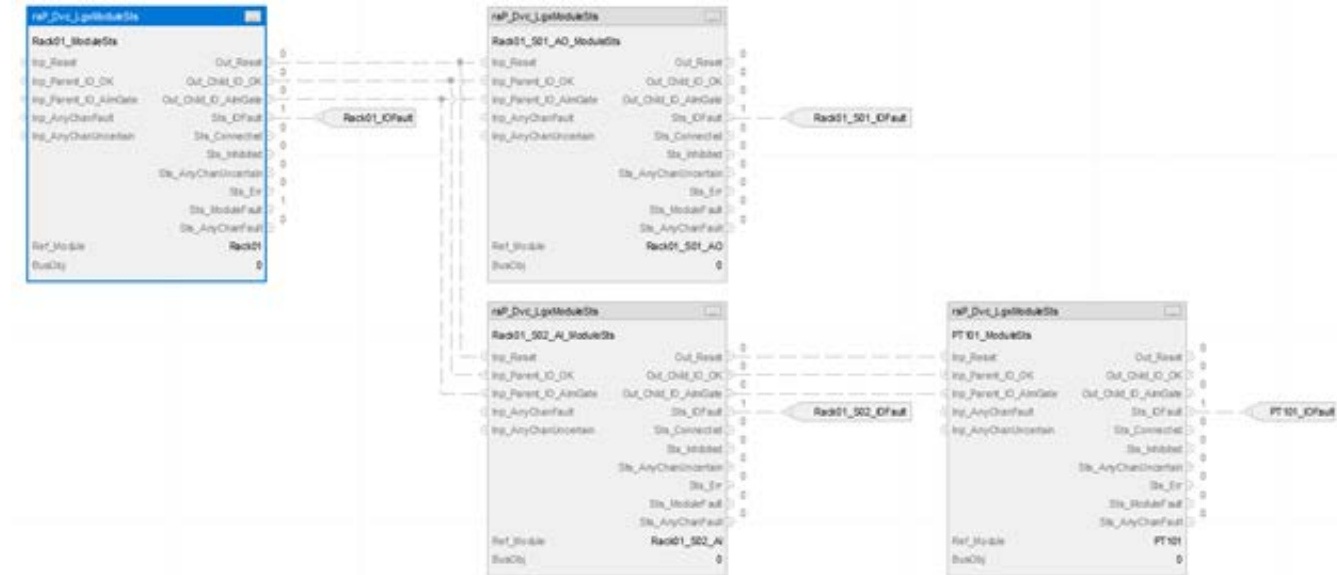
The raP_Dvc_LgxModuleSts also has the capability to be organized, via connecting pins or via the optional Bus, to match the I/O hierarchy. This organization can help prevent alarm floods by inhibiting the alarms from lower-level modules when a higher-level connection fault causes a cascading I/O failure.

Using the following I/O configuration as an example:



If the connection to the 5094-AENTR adapter fails, all devices under it will report I/O connection loss, and a flood of Module Fault alarms occur. By wiring the raP_Dvc_LgxModuleSts instructions into a hierarchy, the fault that is detected for the 5094-

AENTR can be cascaded to the I/O Fault inputs of all dependent devices, AND can be used to inhibit the Module Fault alarms at the lower-levels, reducing the number of alarms generated.



If the I/O adapter (5094-AENTR) connection fails, the HART analog input module (5094-IF8IH) and the PT101 device logic will be informed of the I/O fault condition (via the Rack01_S02IOfault and PT101_IOfault IREFs), but the analog input module and PT101 device Module Fault alarms are inhibited (via the child I/O alarm gate connections), so that only the root cause module fault alarm will be generated.

Logix Task Monitor (raP_Dvc_LgxTaskMon)

The raP_Dvc_LgxTaskMon (Logix Task Monitor) Add-On Instruction monitors one task running in a Logix controller to provide task statistics, such as task scan time and overlap count.

The instruction also provides the following:

- Display Values for Task configuration: Task Name, Priority, Rate, Watchdog settings
- Display Values for Task statistics: Last and Max Execution time, Overlap Count
- Display of the Task Inhibit / Active Status
- Configuration of a "Plan" Execution Time, the time in which the Task is expected to always complete (including interrupts by higher priority Tasks)
- An optional Alarm when Last (actual) Execution Time exceeds Plan
- A Reset command, which clears and acknowledges the Over Plan Alarm.
- Maintenance Commands to clear the Max Execution Time and Overlap Count.

Guidelines

Use this instruction in these situations:

- Monitor the execution of one or more tasks in a Logix controller
- Set an alarm when task execution time exceeds a 'plan' threshold

Do **not** use this instruction if you are using suitable software or another method to monitor controller task execution.

Functional Description

The raP_Dvc_LgxTaskMon instruction includes an Add-On Instruction for use with:

- Studio 5000 Logix Designer software, version 33 or later

- Logix controllers, firmware revision 33 or later
- Graphic symbol and faceplate display for use with FactoryTalk® View Site Edition software, version 13 or later.

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 Files

The raP_Dvc_LgxTaskMon_5.30.**00**_A0I.L5X or (raP_Dvc_LgxTaskMon_5.20.**00**_A0I.L5X) Add-On Instruction must be imported into the controller project to be able to be used in the controller configuration. The maintenance release number (boldfaced) can change as maintenance revisions are created.

Visualization Files

See [Library Versions on page 15](#) for general information on visualization files.

Operations

The raP_Dvc_LgxTaskMon Add-On Instruction does not use modes or virtualization.

Command Sources

The raP_Dvc_LgxModuleSts instruction has no commands or outputs that are intended to control equipment and therefore does not have any selection of active command source.

Alarms

The raP_Dvc_LgxTaskMon Instruction uses the following alarms, which are implemented by using Tag Based Alarms.

Alarm	Alarm Name	Description
Over Plan	Alm_OverPlan	Task execution time exceeds plan limit.

Virtualization

Virtualization allows the raP_Dvc_LgxModuleSts instruction to report a virtual connection status for use in test, demonstration, or training systems. The raP_Dvc_LgxModuleSts Add-On Instruction can be selected to virtual or physical (normal) operation. When physical operation is selected, the actual module connection status is monitored, and an I/O Fault status and Module Fault alarm is reported if the connection is not running. When virtual operation is selected, the actual module connection status is ignored; the Set_VirtualConnectedSts input parameter determines the reported connection status.

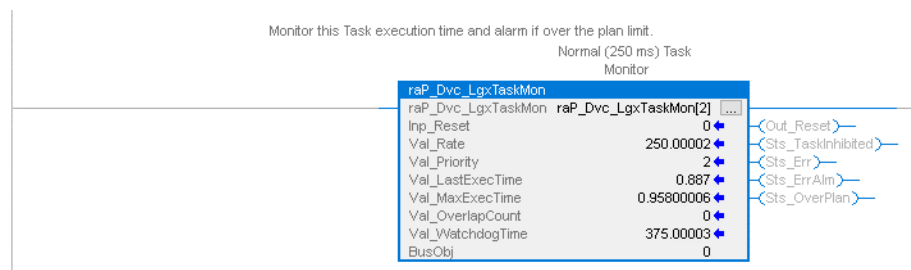
Set_VirtualConnectedSts value	Description
1	Connected, the connection status is reported as OK
0	Faulted, the connection status is reported as faulted, the Sts_IOFault status is raised for dependent devices, and the Alm_ModuleFault alarm is raised.

Tag Extended Properties and Default Alarm Settings

Common	
raP_Dvc_LgxTaskMon.@Description	"Logix Task Monitor"
raP_Dvc_LgxTaskMon.@Area	"Area01"
raP_Dvc_LgxTaskMon.@Instruction	"raP_Dvc_LgxTaskMon"
raP_Dvc_LgxTaskMon.@Label	"Logix Task Monitor"
raP_Dvc_LgxTaskMon.@Library	"raP-5_30"
raP_Dvc_LgxTaskMon.@URL	"n/a"

Alarms		Alarm Default Message	Severity
raP_Dvc_LgxTaskMon.Sts_OverPlan.@Label	"Task scan time over plan"	Controller Task /*S:0 %Tag1*/: execution time over plan Tag1 = raP_Dvc_LgxTaskmon.Sts_sName (Task Name as configured in the controller.):STRING	500

Programming Example



Place an instance like this in each controller task and have it always scanned true. In ladder diagram, you can use an array of backing tags, assigning a different array member for each task. For the default PlantPax[®] process controller task model, the following assignments are recommended and are included in the PlantPax template application:

raP_Dvc_LgxTaskMon[0]	System (1000 ms) task
raP_Dvc_LgxTaskMon[1]	Fast (100 ms) task
raP_Dvc_LgxTaskMon[2]	Normal (250 ms) task
raP_Dvc_LgxTaskMon[3]	Slow (500 ms) task

The task in which the instruction is instantiated will be checked each scan for its status and to determine whether its execution time exceeds the plan limit for that task.

Logix Event (raP_Tec_LgxEvent)

The raP_Tec_LgxEvent (Logix Event) Add-On Instruction captures any of 16 event bit rising edge transitions and records the lowest-order rising edge bit as the reason of the event. The instruction provides an "I/O fault" input to monitor parent I/O conditions. It also provides a Reset to clear the event reason.

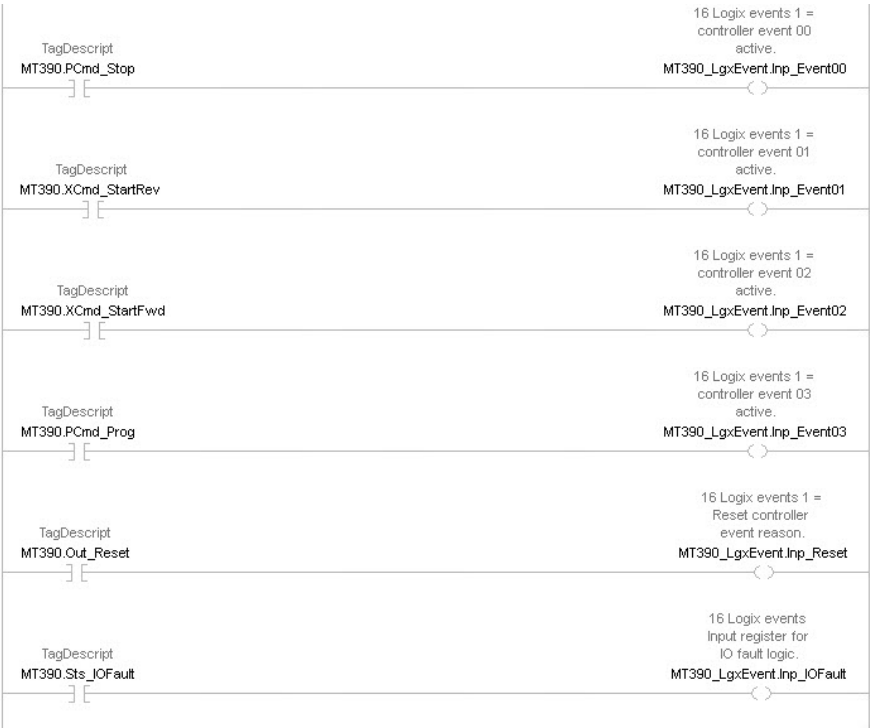
Guidelines

Use this instruction if you want to monitor up to 16 User-defined events, per object.

Functional Description

The raP_Tec_LgxEvent Add-On Instruction is used to capture any of 16 bit rising edge transitions and records the lowest order rising edge bit as the reason of the event. The events are published in the Sts.Reasons parameter. Reasons are only captured if the Inp_IOFault and the Inp_Reset parameters are low. (0) Event reasons are cleared by setting the Reset input parameter. (1) Event input conditions can be connected to any user-defined logic.

The following images show how the event inputs are mapped to the instruction. Ladder logic is typically used to allow for more complex trigger conditions. Here is the code showing the mapping of four event triggers, as well as the Reset and IOFault:



Those inputs are the source of the raP_Tec_LgxEvent instruction:



The following images show how the event Reasons are assigned a User description for the Event. Each individual event is allowed a unique description to be applied. The description must be changed/updated to what the you want to be displayed in the reports.

Name	Usage	Value	Force Mask	Style	Data Type	Description
raP_Tec_LgxEvent_01.Sts_Reason		2#0000_0000_0000_00...		Binary	INT	16 Logix events Reason of individual controller event.
raP_Tec_LgxEvent_01.Sts_Reason.0		0		Decimal	BOOL	16 Logix events Reason Zero
raP_Tec_LgxEvent_01.Sts_Reason.1		0		Decimal	BOOL	16 Logix events Reason One
raP_Tec_LgxEvent_01.Sts_Reason.2		0		Decimal	BOOL	16 Logix events Reason Two
raP_Tec_LgxEvent_01.Sts_Reason.3		0		Decimal	BOOL	16 Logix events Reason Three
raP_Tec_LgxEvent_01.Sts_Reason.4		0		Decimal	BOOL	16 Logix events Reason Four
raP_Tec_LgxEvent_01.Sts_Reason.5		0		Decimal	BOOL	16 Logix events Reason Five
raP_Tec_LgxEvent_01.Sts_Reason.6		0		Decimal	BOOL	16 Logix events Reason Six
raP_Tec_LgxEvent_01.Sts_Reason.7		0		Decimal	BOOL	16 Logix events Reason Seven
raP_Tec_LgxEvent_01.Sts_Reason.8		0		Decimal	BOOL	16 Logix events Reason Eight
raP_Tec_LgxEvent_01.Sts_Reason.9		0		Decimal	BOOL	16 Logix events Reason Nine
raP_Tec_LgxEvent_01.Sts_Reason.10		0		Decimal	BOOL	16 Logix events Reason Ten
raP_Tec_LgxEvent_01.Sts_Reason.11		0		Decimal	BOOL	16 Logix events Reason Eleven
raP_Tec_LgxEvent_01.Sts_Reason.12		0		Decimal	BOOL	16 Logix events Reason Twelve
raP_Tec_LgxEvent_01.Sts_Reason.13		0		Decimal	BOOL	16 Logix events Reason Thirteen
raP_Tec_LgxEvent_01.Sts_Reason.14		0		Decimal	BOOL	16 Logix events Reason Fourteen
raP_Tec_LgxEvent_01.Sts_Reason.15		0		Decimal	BOOL	16 Logix events Reason Fifteen

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 Files

The raP_Tec_LgxEvent_5.30.**00**_A0I.L5X or (raP_Tec_LgxEvent_5.20.**00**_A0I.L5X) Add-On Instruction definition file must be imported into the controller project to be able to be used in the controller configuration. The maintenance release number (boldfaced) can change as maintenance revisions are created.

Visualization Files

There are no visualization files because the raP_Tec_LgxEvent object does not use Graphic Symbols or Faceplates.

Operations

Command Sources

The raP_Tec_LgxEvent instruction has no commands or outputs that are intended to control equipment and therefore does not have any selection of active command source.

Alarms

The raP_Tec_LgxEvent Instruction has no Alarms.

Virtualization

The raP_Tec_LgxEvent Instruction has no Virtualization.

Execution

The following table explains the handling of instruction execution conditions.

Condition	Description
EnableIn False (false rung)	No EnableIn False logic is provided. The raP_Tec_LgxEvent instruction must always be scanned true. In relay ladder logic, the raP_Tec_LgxEvent instruction must be by itself on an unconditional rung.
Powerup (prescan, first scan)	No SFC Prescan logic is provided.
Postscan (SFC transition)	No SFC Postscan logic is provided.

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

Programming Examples

The example in the Function Description section shows the basic use of the raP_Tec_LgxEvent Add-On Instruction for capturing events.

Process Extended Alarms (raP_Opr_ExtddAlm)



For the object and visualization parameters, see PlantPAx Process Objects, publication [PROCES-RD200](#), and PlantPAx Visualization Files, publication [PROCES-RD201](#).

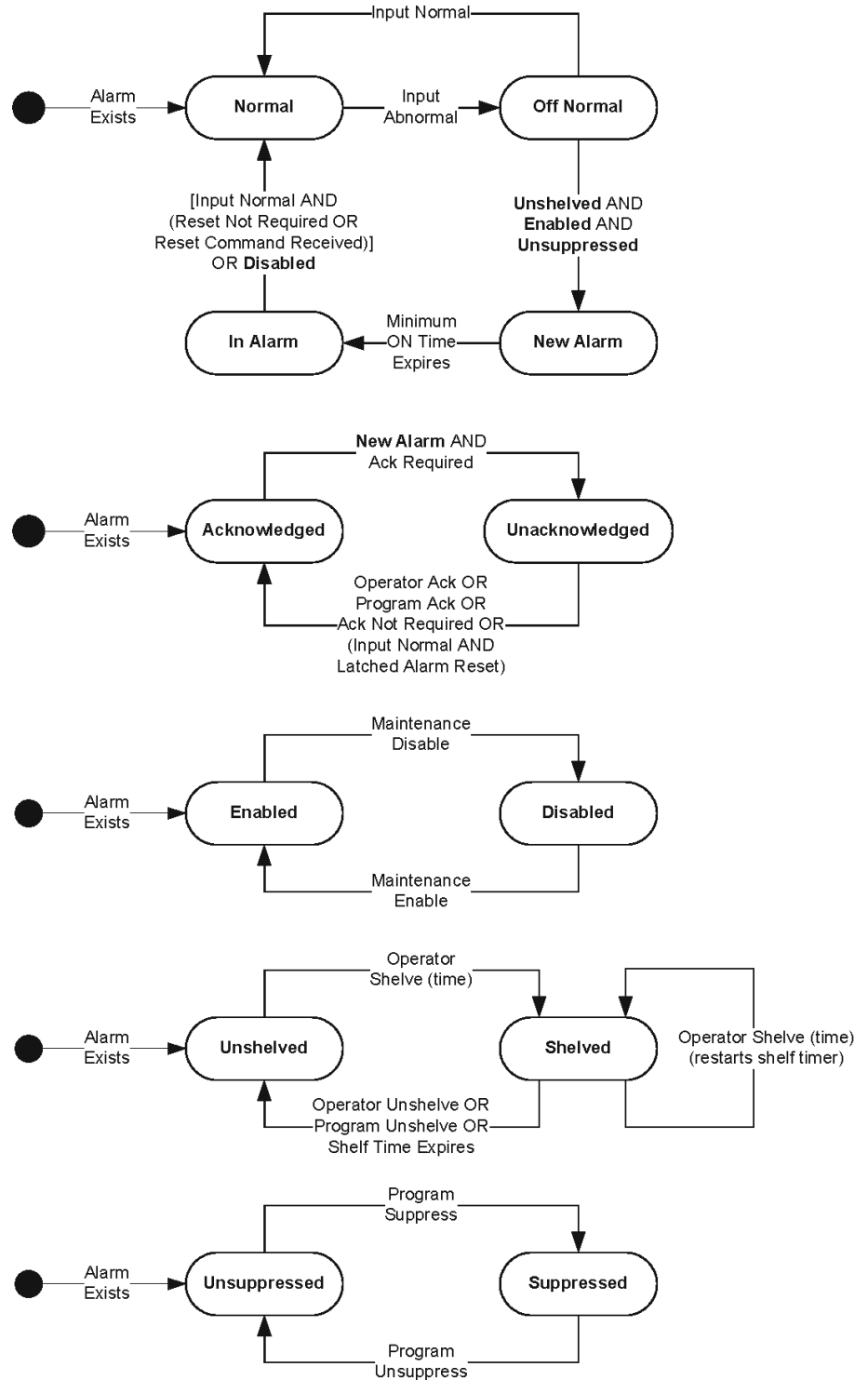
The raP_Opr_ExtddAlm (Extended Alarm Block) Add-On Instruction is used to provide notification to operators of abnormal conditions or events for up to 32 additional items external to a parent object. (raP_Opr_Area, raP_Opr_Unit, raP_Opr_EMGen, raP_Opr_EPGen)
This instruction handles the connections of the commands from the parent object:

- Acknowledge
- Reset
- Enabling/Disabling
- Suppress/Unsuppress
- UnShelve

This instruction handles the connections of the status from the raP_Opr_ExtddAlm:

- Used
- Alarm
- Acknowledged
- Disabled
- Suppressed
- Shelved
- Alarm Fault
- Ready for Reset
- Notify value

The state diagram shows how a raP_Opr_ExtddAlm 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 raP_Opr_ExtddAlm instruction include the following:

- Raise an alarm when the input is true.
- Handle Alarm Acknowledge commands from the HMI or from the parent object. The requirement for acknowledgment is configurable. If acknowledgment is required, a new alarm clears the acknowledged status and an Acknowledge command is required to set the status. If acknowledgment is not required, the alarm is automatically acknowledged.
- Handle Alarm Reset commands from the HMI or from the parent object. 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.
- Handle Maintenance Disable and Enable commands, Program Suppress and Unsuppress commands, and Operator Shelve and Unshelve commands. 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.

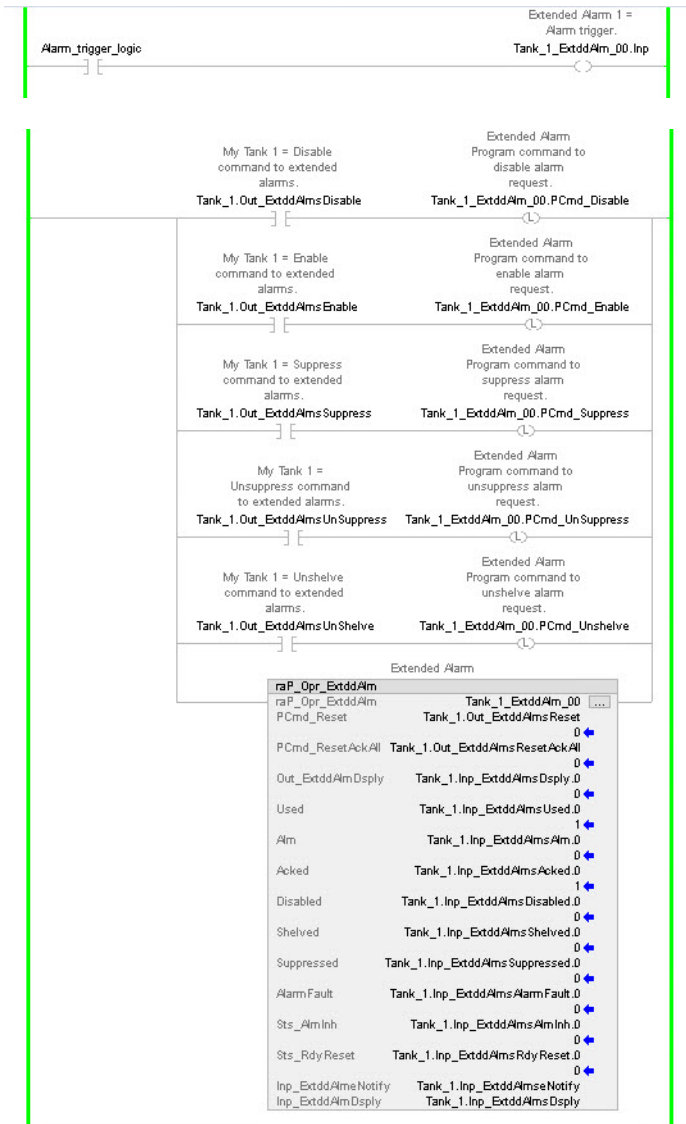
When an alarm is Disabled by Maintenance, the following occurs:

- 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.

The following images show how the raP_Opr_ExtddAlm is connected to the first allowable instance of a parent extended alarm. The first instance is mapped to the bit level of a DINT, the first is bit 0, the last, bit 31.



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 Files

The `raP_Opr_ExtddAlm_5.30.00.A01.L5X` or (`raP_Opr_ExtddAlm_5.20.00.A01.L5X`) Add-On Instruction must be imported into the controller project to be used in the controller configuration. The maintenance release number (boldfaced) can change as maintenance revisions are created.

Visualization Files

See [Library Versions on page 15](#) for general information on visualization files.

Operations

Command Sources

The raP_Opr_ExtdAlm instruction has no commands or outputs that are intended to control equipment and therefore does not have any selection of active command.

Alarms

The raP_Opr_ExtdAlm Instruction uses the following alarm, which is implemented using Logix Tag Based Alarms:

Alarm	Alarm Name	Description
Alarm	Alm_Alarm	Raised when an extended alarm condition is asserted.

Virtualization

The raP_Opr_ExtdAlm Add-On Instruction does not have a virtualization capability.

Execution

The following table explains the handling of instruction execution conditions.

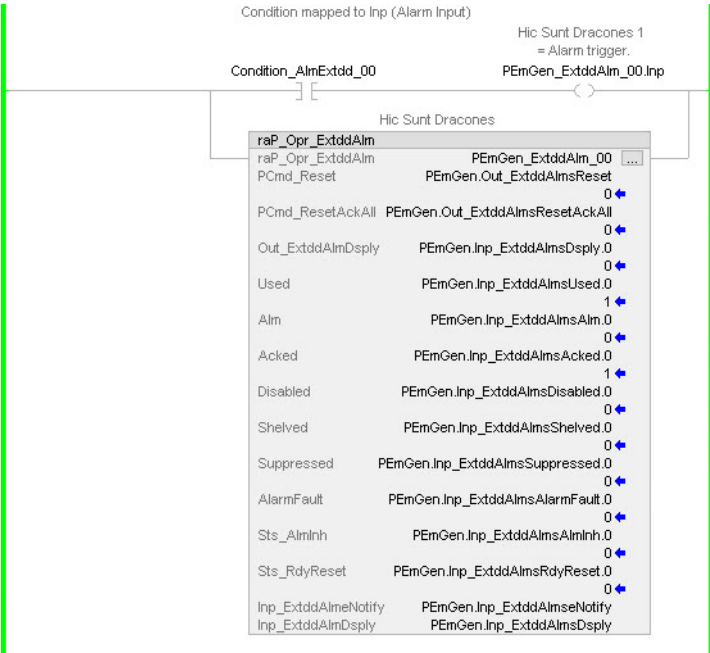
Condition	Description
EnableIn False (false rung)	Processing for EnableIn False (False Rung) is handled the same as the main Logic Routine except that the state of Inp (the Input) is inverted. This inversion lets the raP_Opr_ExtdAlm Add-On Instruction in a ladder diagram instance have its input mapped 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)	All commands, including alarm acknowledge and reset, are cleared on prescan.
Postscan	No SFC Postscan logic is provided.

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

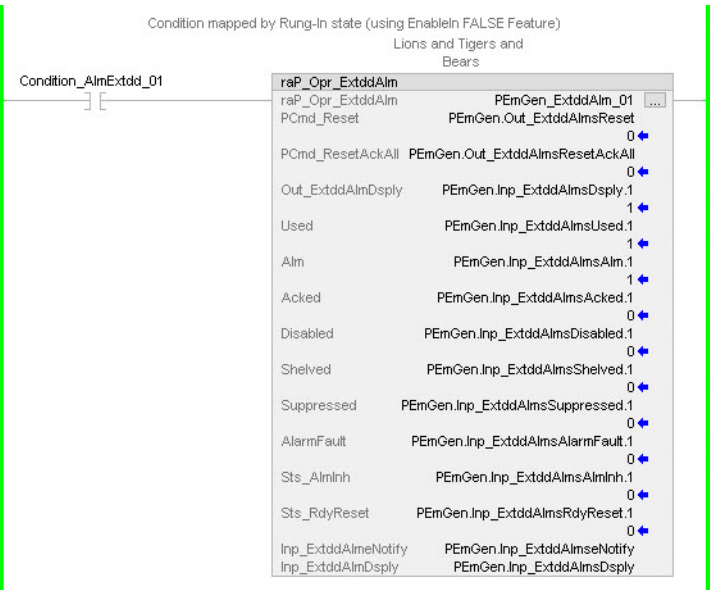
Programming Examples Implementation by Using the EnableIn False Feature

For the convenience of ladder diagram programmers, the raP_Opr_ExtddAlm 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 raP_Opr_ExtddAlm 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 raP_Opr_ExtddAlm 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).

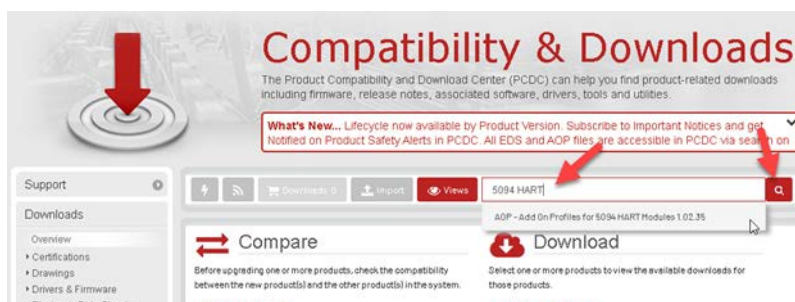
5094-IF8IH to PAH Configuration Example

This appendix describes how to configure a HART device using a newer HART I/O module, such as the 5094-IF8IH, and the PAH instruction, in a PlantPAx® 5.0 system. This example requires a system that meets PlantPAx 5.0 system requirements, including using Version 33 or later of Studio 5000 Logix Designer® software.

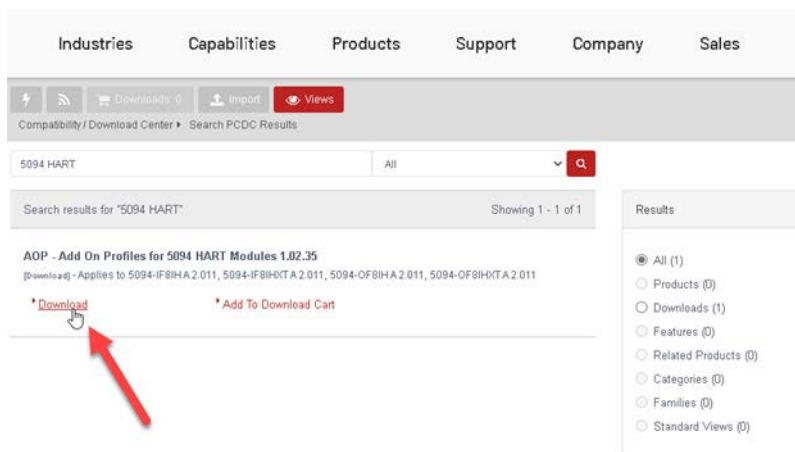
Download and install the 5094 HART Analog Add-On Profile

The Add-on Profile can be accessed from the [Product Compatibility and Download Center](#).

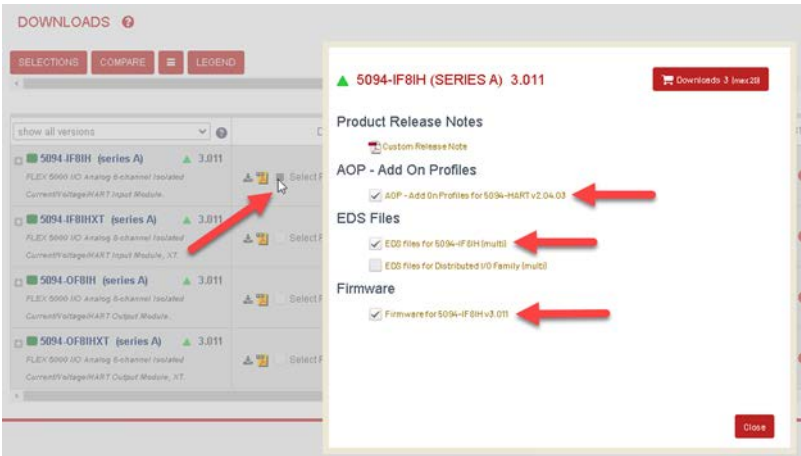
1. Search for "5094-HART".



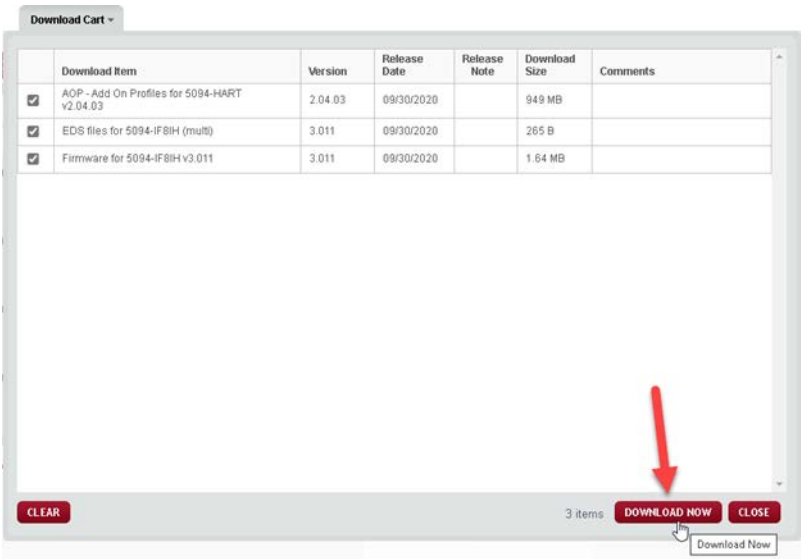
2. Select Download.



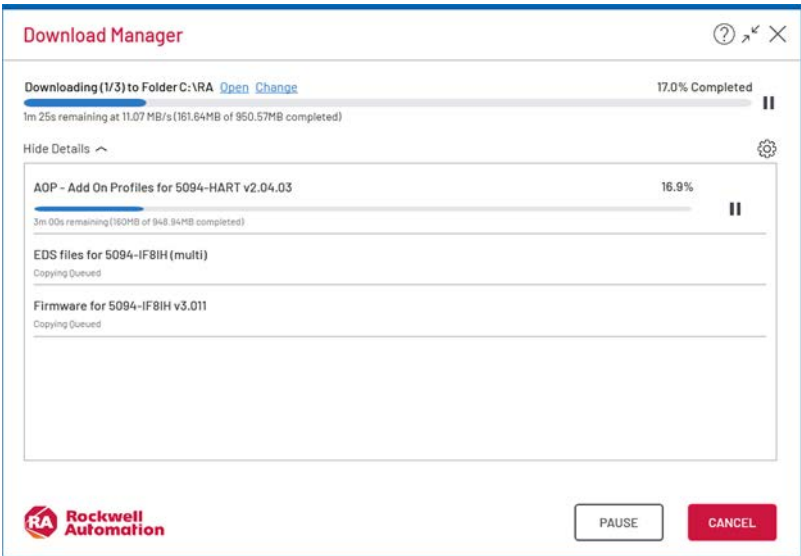
3. Select Files, then select the Add-on Profiles, EDS Files, and Firmware.



4. Select Download Now.



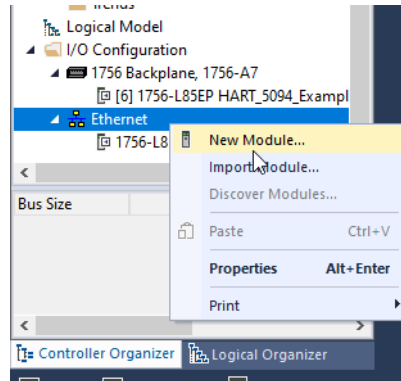
The files are downloaded into a zip file using the download manager.



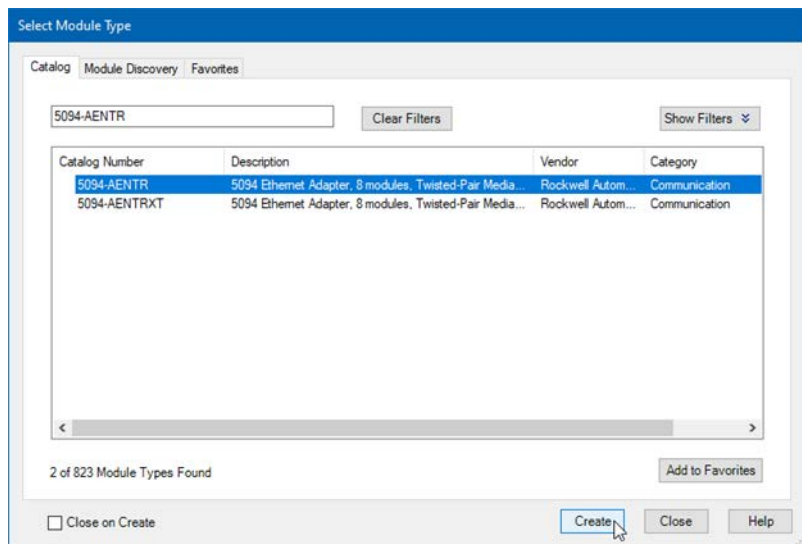
5. Extract the files from the ZIP folder.
6. Run mpsetup.exe as Administrator.

Add the 5094 Adapter Module to the Project I/O Configuration

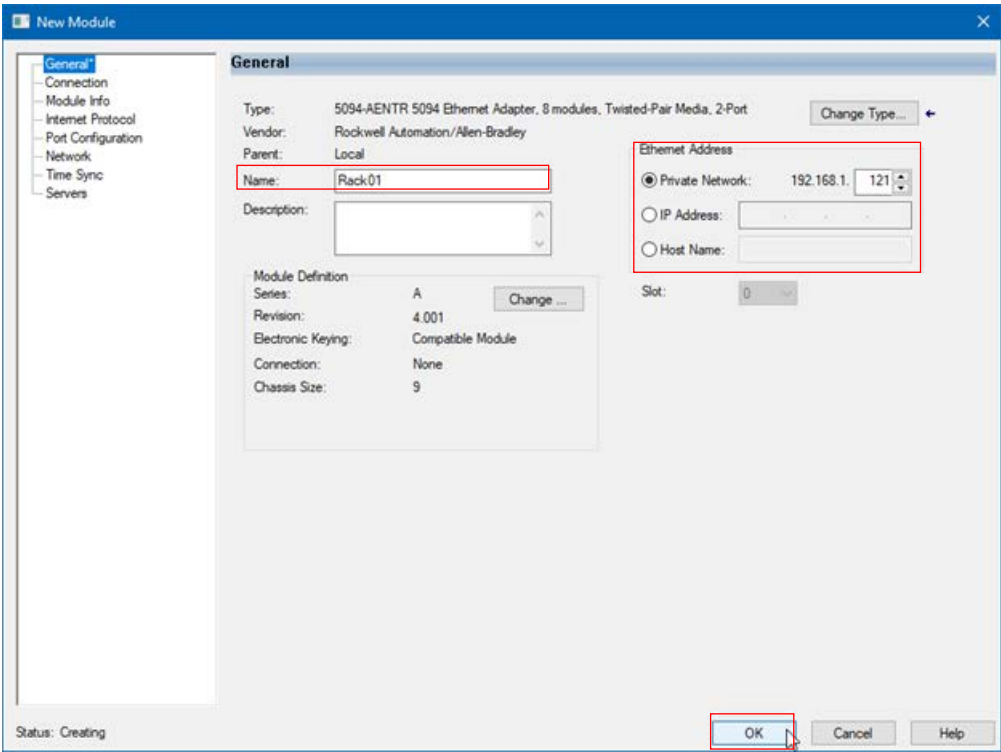
1. In the controller Organizer for your project, select the Ethernet network to be used to communicate with the 5094 I/O. Right-click and select "New Module..."



2. Select the catalog number of the 5094 adapter that you are using and Create.

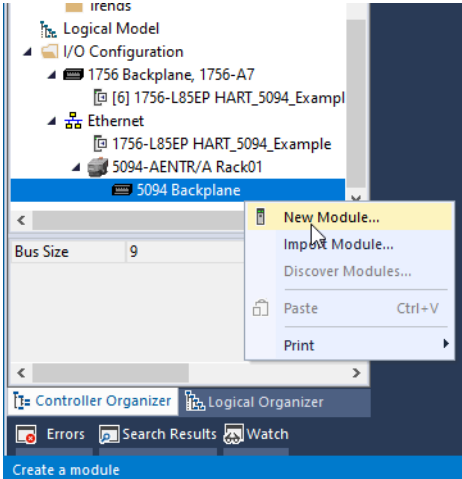


3. Enter the name and IP address for the adapter.

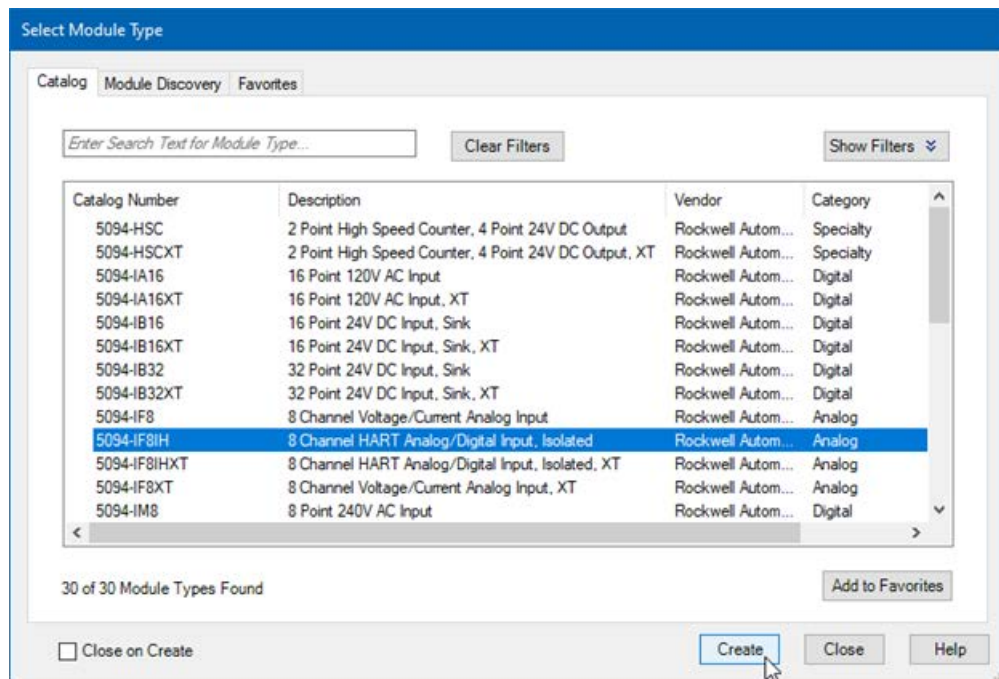


Add the 5094-IF8IH Module to the Project I/O Configuration

- 1. In the controller Organizer for your project, select the 5094 Backplane. Right-click and select "New Module..."



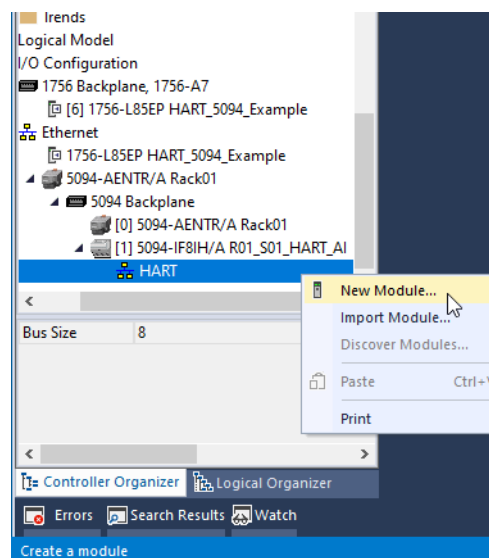
2. Select the 5094-IF8IH module and "Create".



3. To accept the module defaults, Select OK.

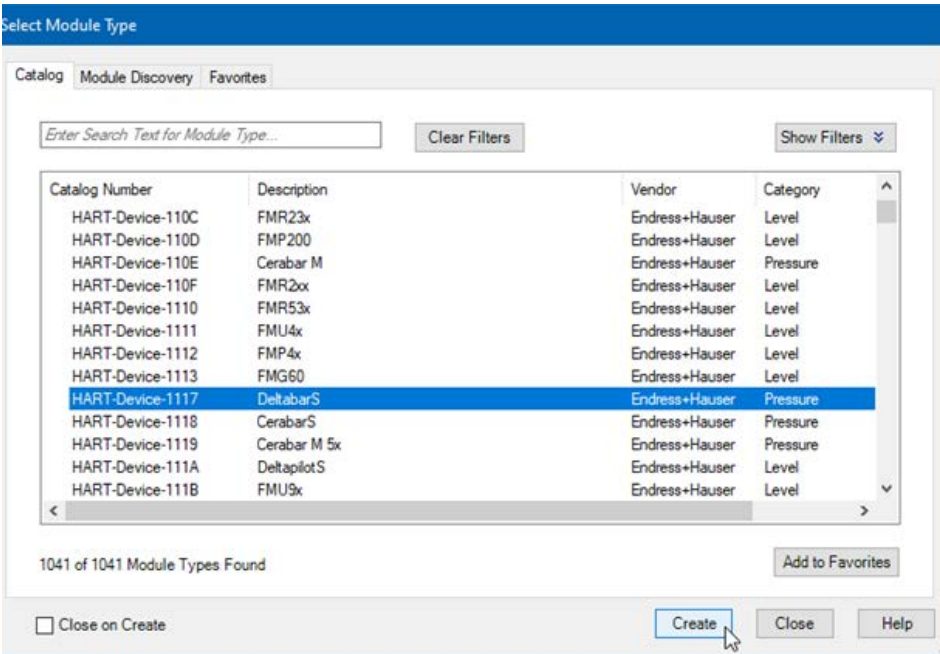
Add the HART Device to the Project I/O Configuration

1. In the controller Organizer for your project, select the HART network. Right-click and select "New Module..."

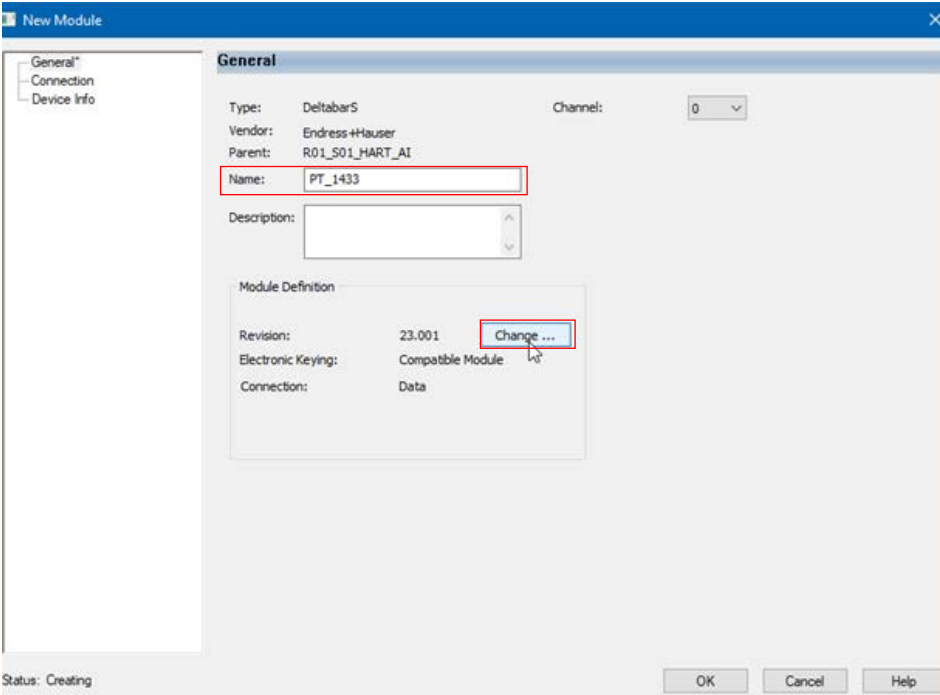


2. Select the type of HART transmitter and "Create".

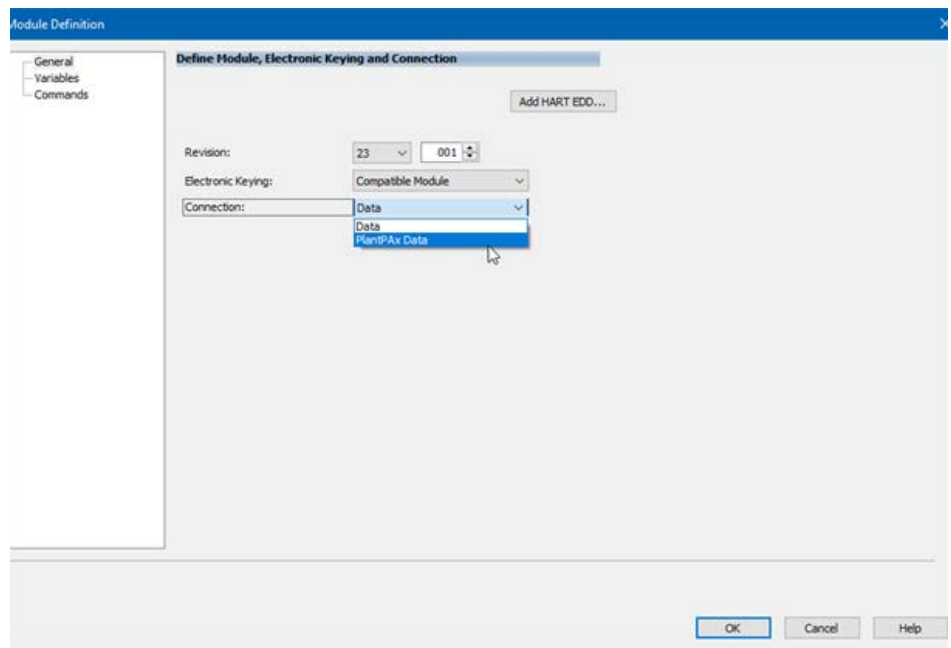
In this example, we are using an Endress+Hauser Deltabar-S device.



3. In the New Module dialog box, enter a name for the transmitter then select Change in the Module Definition section.

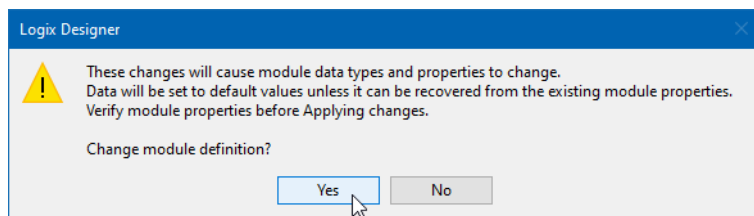


4. Change the Connection type to PlantPAx Data.

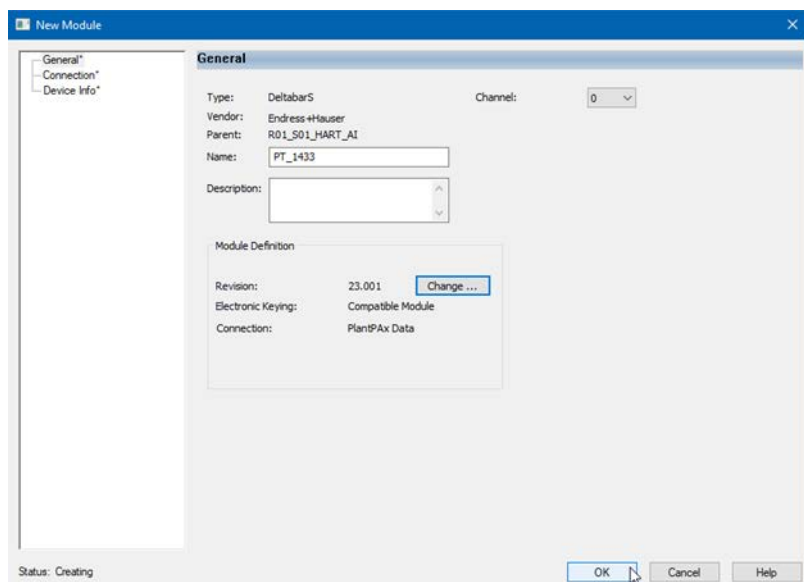


5. Changing the connection type causes a change in data types for the input and output data.

Select Yes to change the module definition.

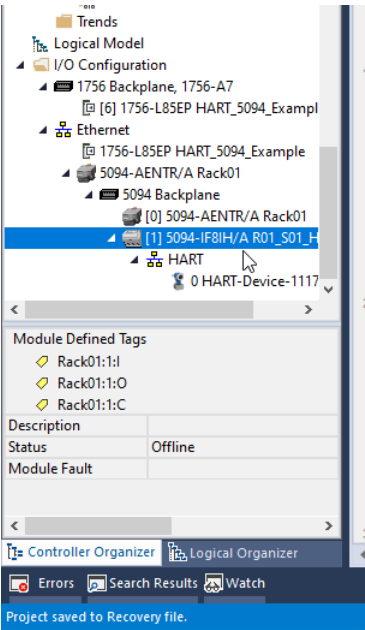


6. Verify the information and Select OK.

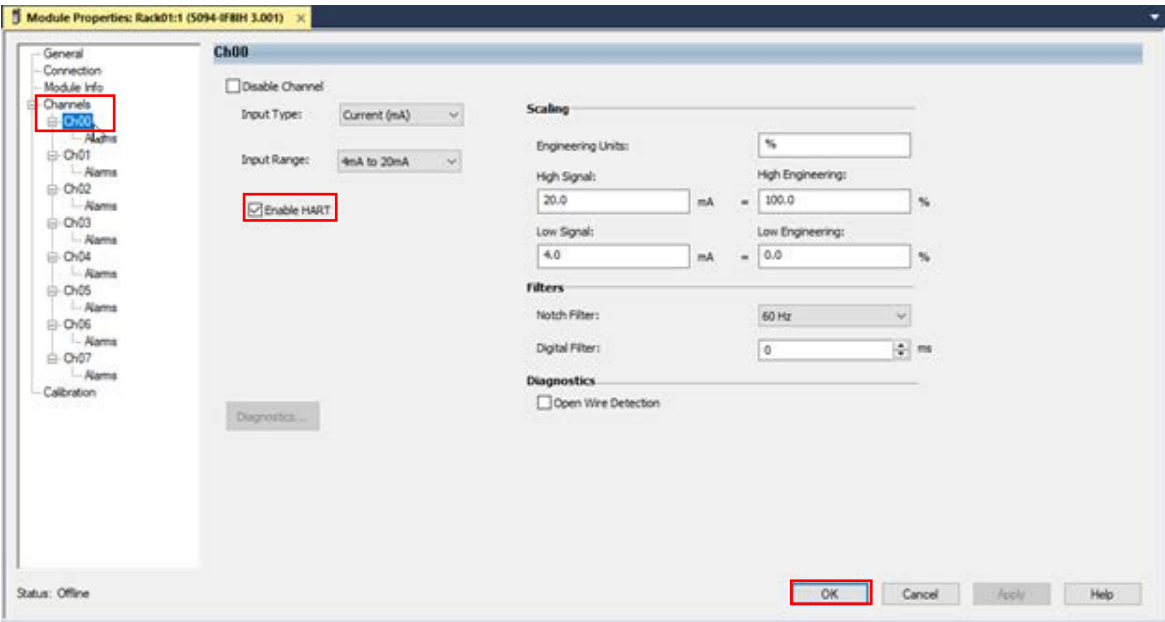


Configure the Analog Input Channel

1. In the controller Organizer for your project, select the 5094-IF8IH module created. Double-click to open the Properties dialog.



2. Select the channel where the transmitter is connected. In this example, it is Channel 00. Select the box to Enable HART communication on this channel.

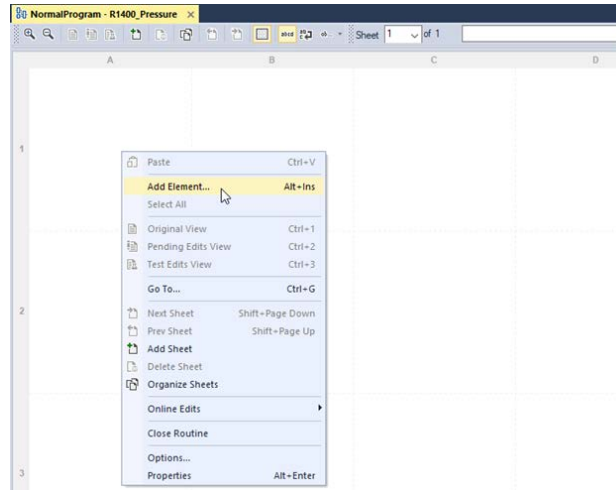


Add the PAH (Process Analog HART) and PAI (Process Analog Input) Instruction Instances to the Project

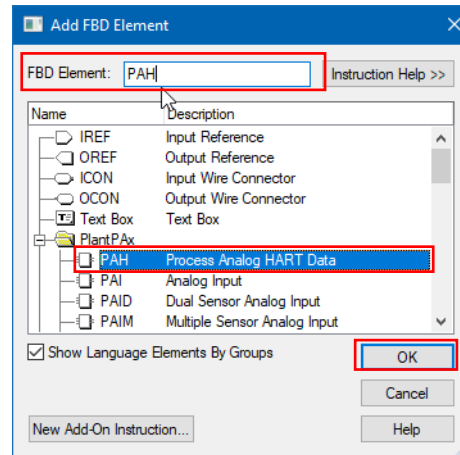
In this example, we are using a Function Block routine. Ladder Diagram or Structured Text could be used.

Add the PAH Instruction Instance

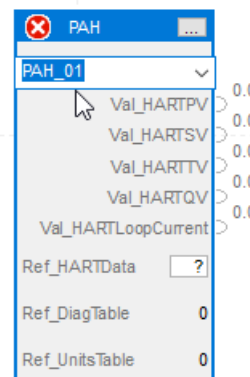
1. Right-click a blank area on the sheet and select "Add Element..."



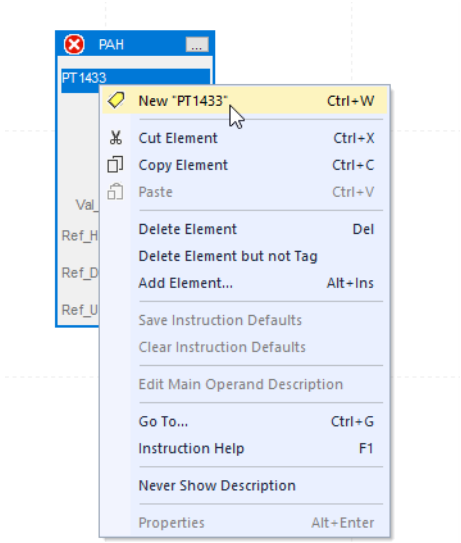
2. Enter PAH for the FBD Element.



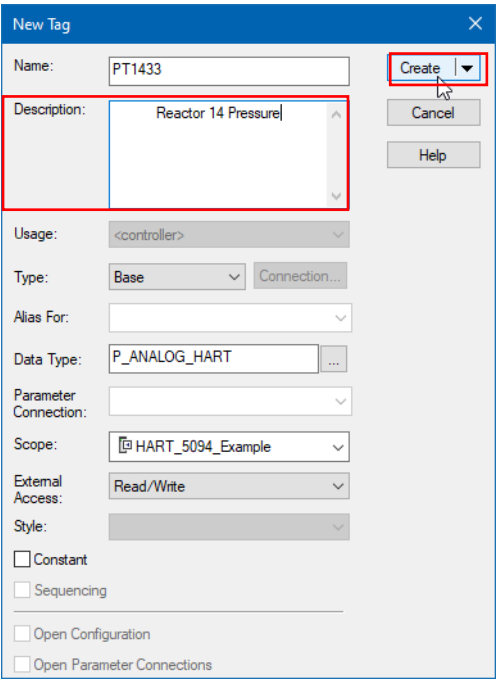
3. Enter the desired tag name of the backing tag for the PAH block.



4. Right-click the new tag name and select "New <tagname>".



5. In the New Tag dialog, enter a tag description. The required data type is automatically selected for you.

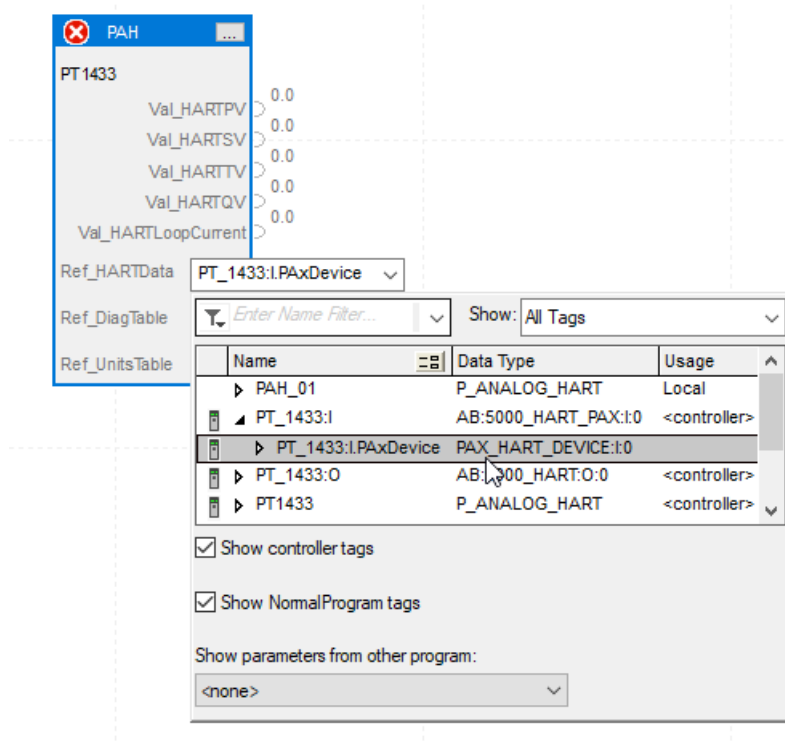


The tag can be created at Controller scope, or in the Program containing this routine. For this example, we use a Controller-scope tag.

Connect PAX_HART_DEVICE:I:0 Member from Input Assembly to Ref_HARTData InOut Parameter.

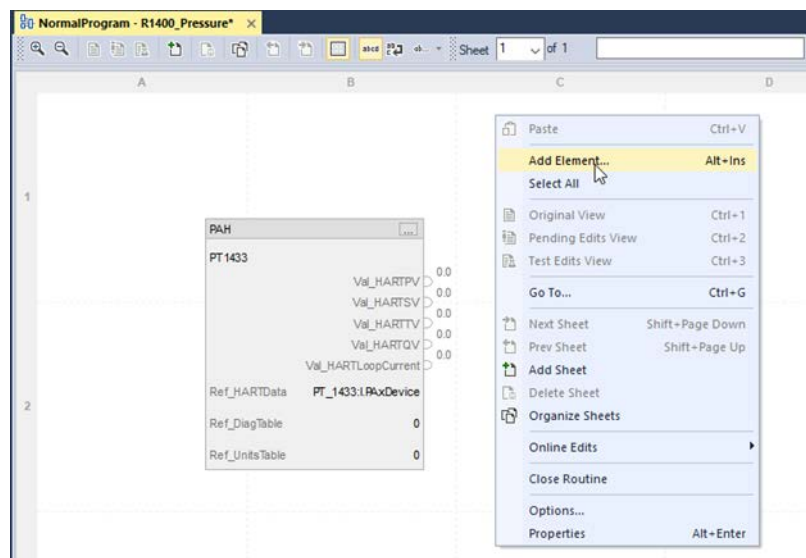
1. Select the dropdown for the Ref_HARTData InOut Parameter.
2. Navigate to the input assembly tag for the HART device, expand, and select the "PaxDevice" member.

The data type must be "PAX_HART_DEVICE:I:0".

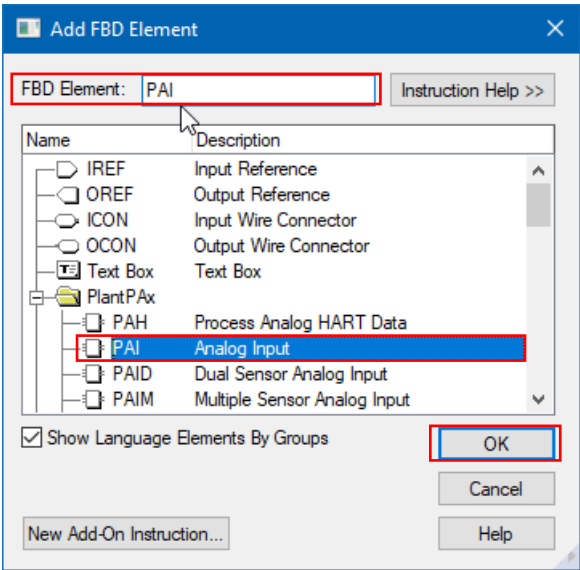


Add the PAI Instruction Instance

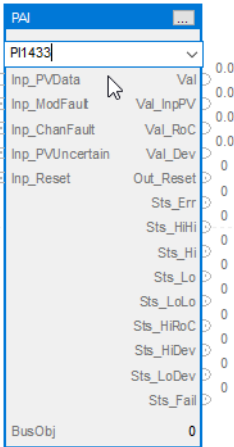
1. Right-click a blank area of the Function Block routine sheet and select "Add Element..."



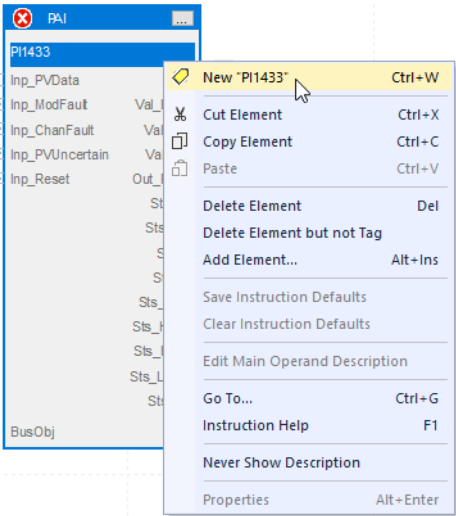
2. Enter PAI for the FBD Element.



3. Enter the desired tag name of the backing tag for the PAI block. In this example, we used "PI1433" (for Pressure Indicator).



4. Right-click the new tag name and select "New <tagname>".

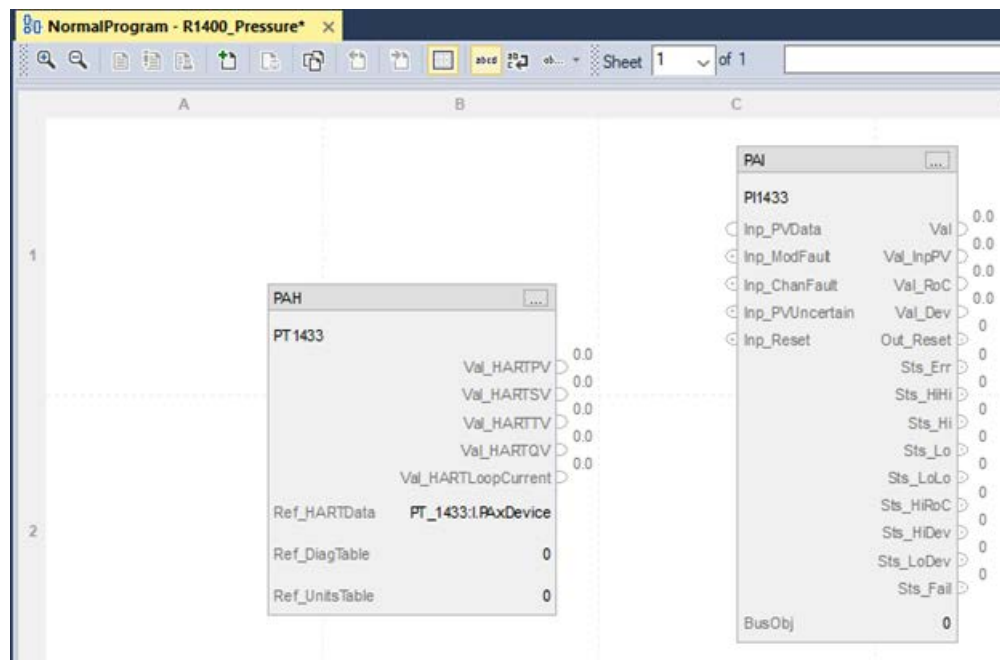


5. In the New Tag dialog, enter a tag description. The required data type is automatically selected for you.



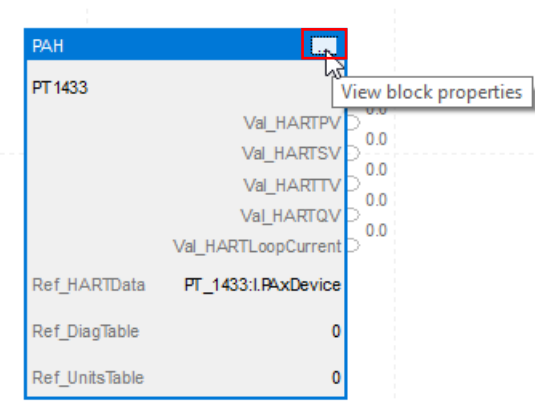
The tag can be created at Controller scope, or in the Program containing this routine. For this example, we use a Controller-scope tag. For HMI navigation to work properly, the PAH and PAI instance tags must be at the same scope.

The tag is created and the routine contains no errors.

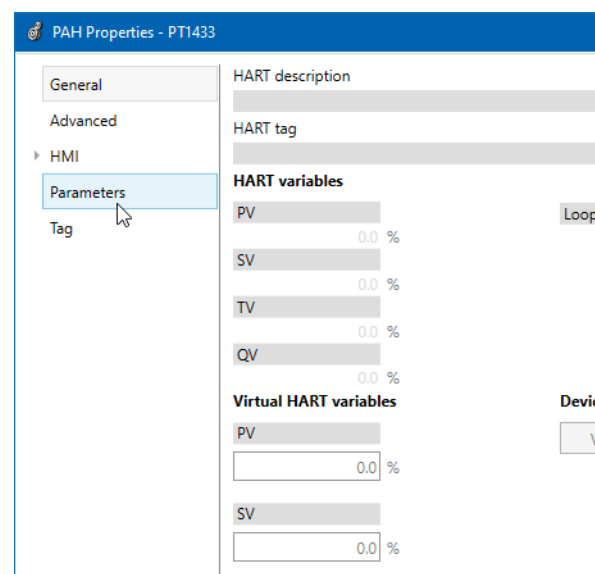


Connect the PAH Instance to the PAI Instance

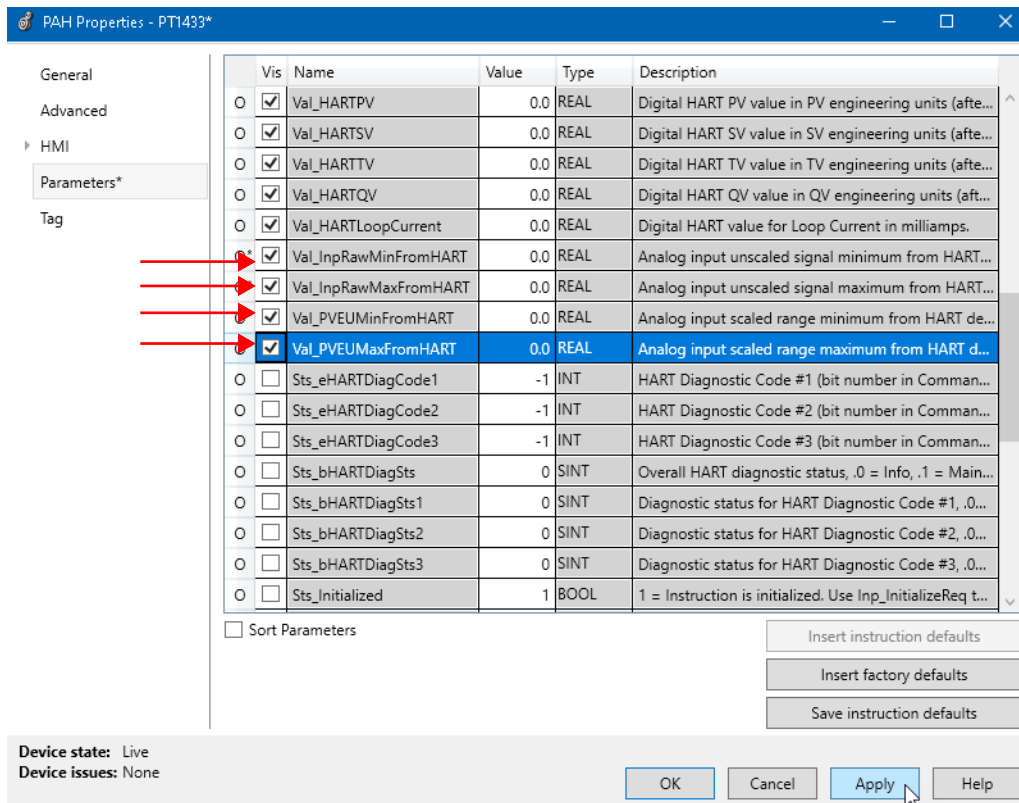
- 1. Select the properties of the PAH instruction.



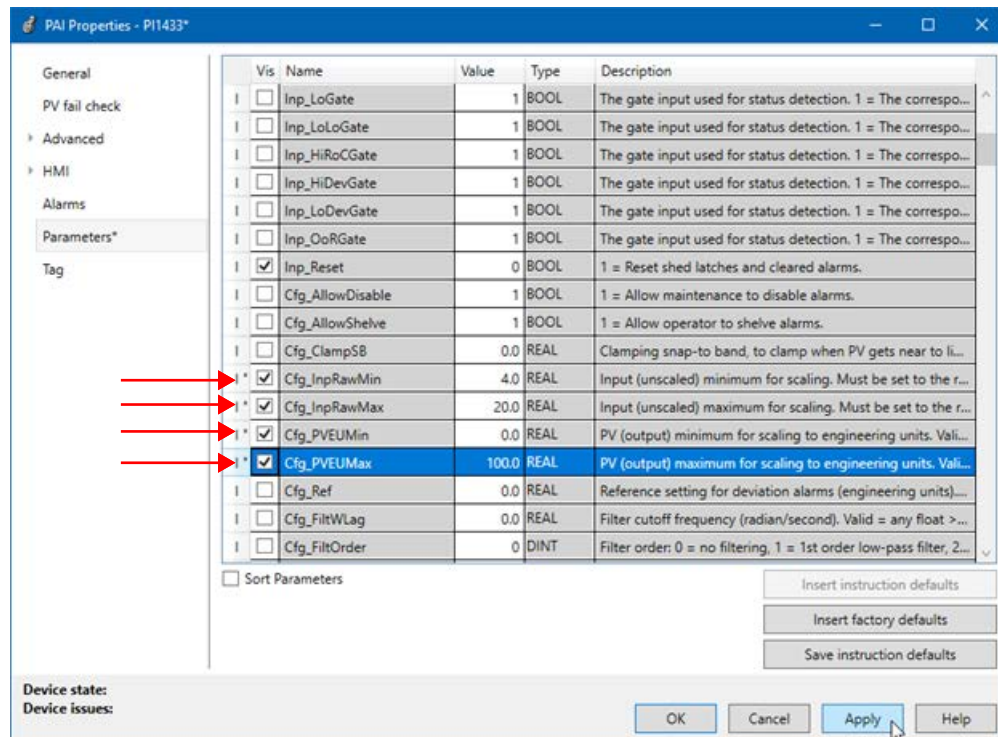
- 2. Select the Parameters tab.



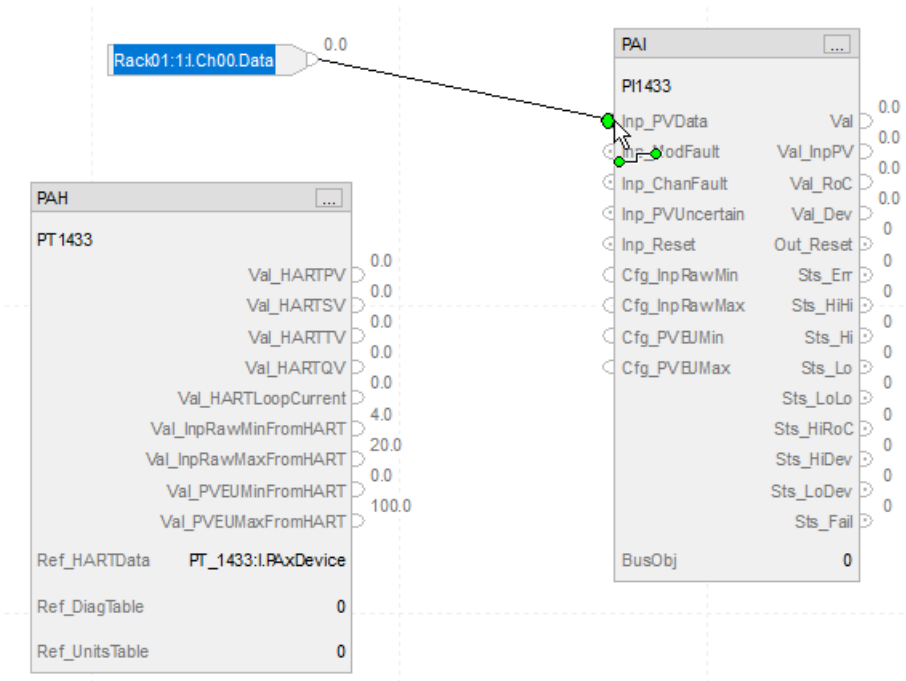
3. Select the boxes in the “Vis” column to make the Raw and EU scaling Values visible as output pins.



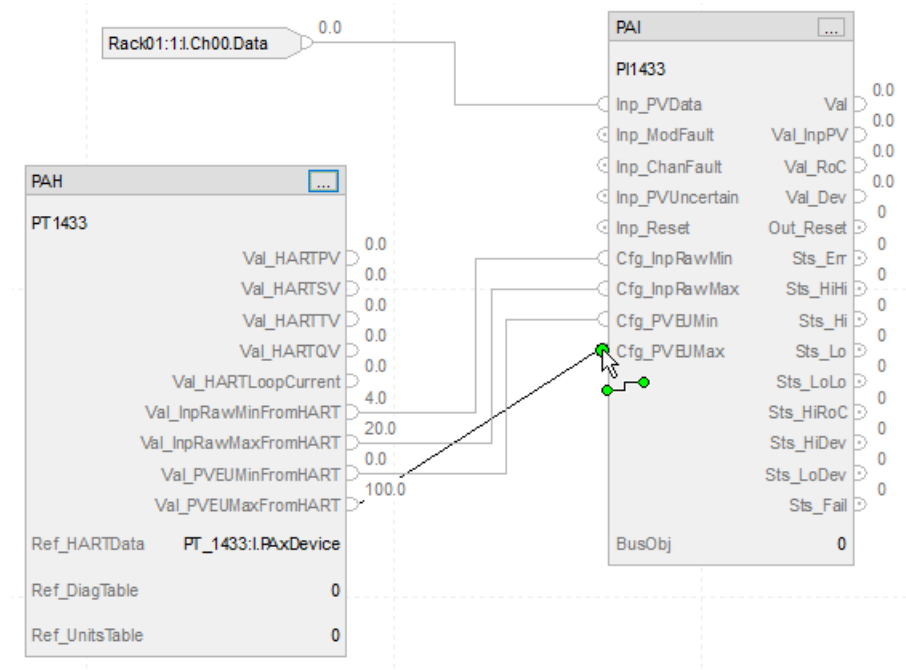
4. From the Parameters tab of the **PAI instruction** properties, Select the boxes in the “Vis” column to make the Raw and EU **configuration input** pins visible.



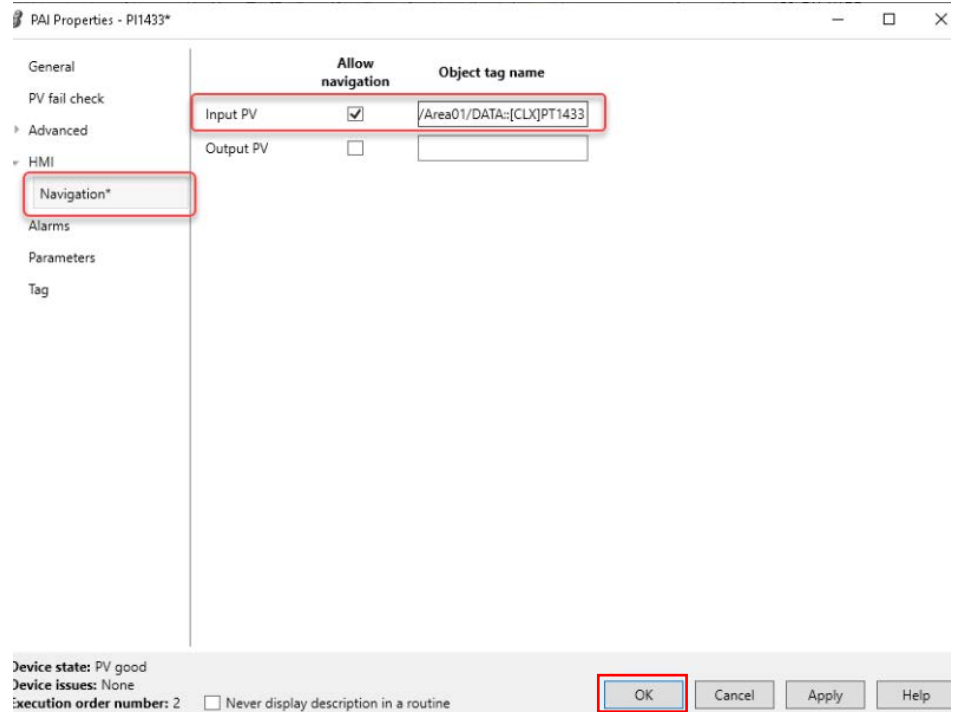
5. Wire the analog input signal to the PAI block.



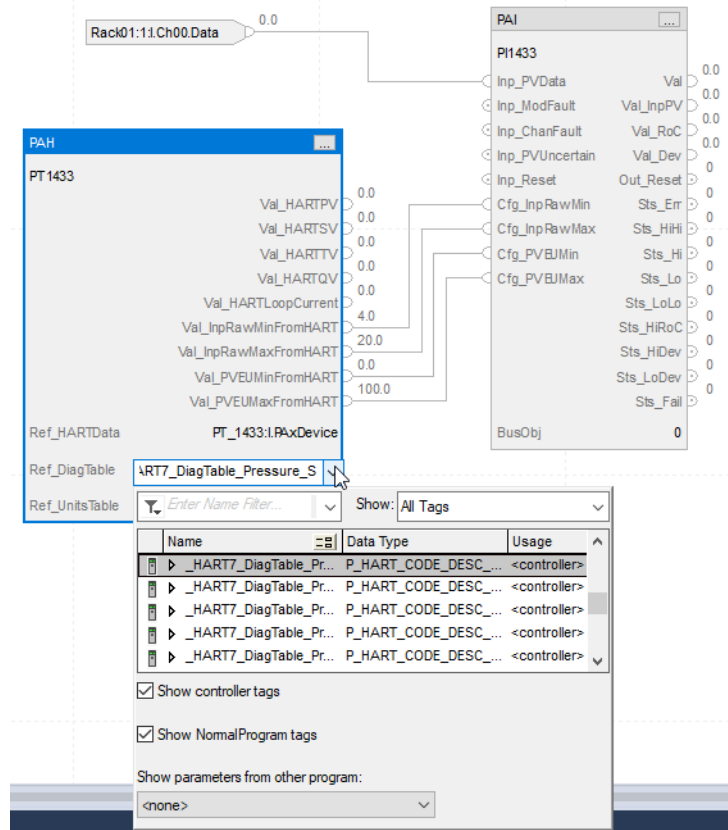
6. Wire the HART scaling data from the PAH block to the PAI block.



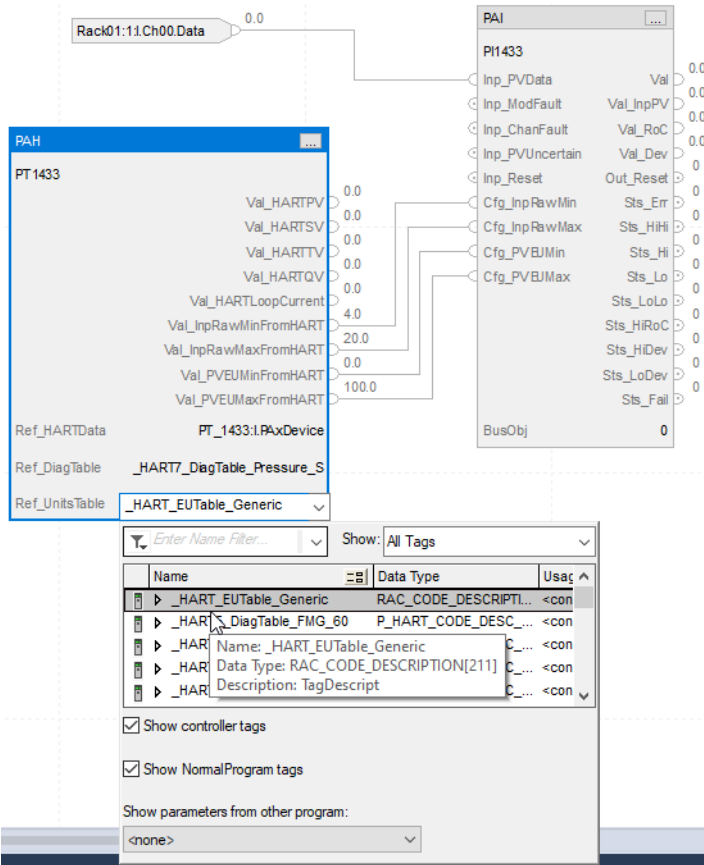
7. From the PAI Properties, navigate to the HMI>Navigation tab. Link the PAI Input PV navigation to the PAH instance.



8. On the PAH instance, link the HART Diagnostic Lookup Table for the pressure transmitter InOut parameter.



9. Link the HART engineering units lookup table to the units InOut parameter.

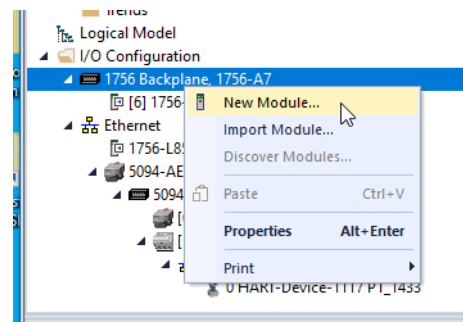


1756-IF8IH with raP_Tec_HARTChanData_to_PAH Add-On Instruction Configuration Example

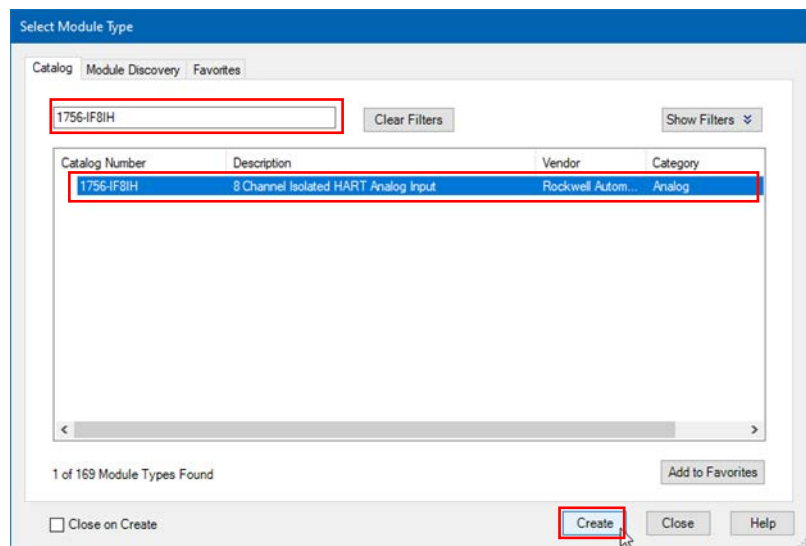
This appendix shows an example of using a 1756-IF8IH (using I_1756IF8IH 4.10) with the raP_Tec_HARTChanData_to_PAH Add-On Instruction from the 5.00 or later Library download to feed PAH and PAI instructions (5.00 or later system on L85EP).

Add the 1756-IF8IH Module to the Project I/O Configuration

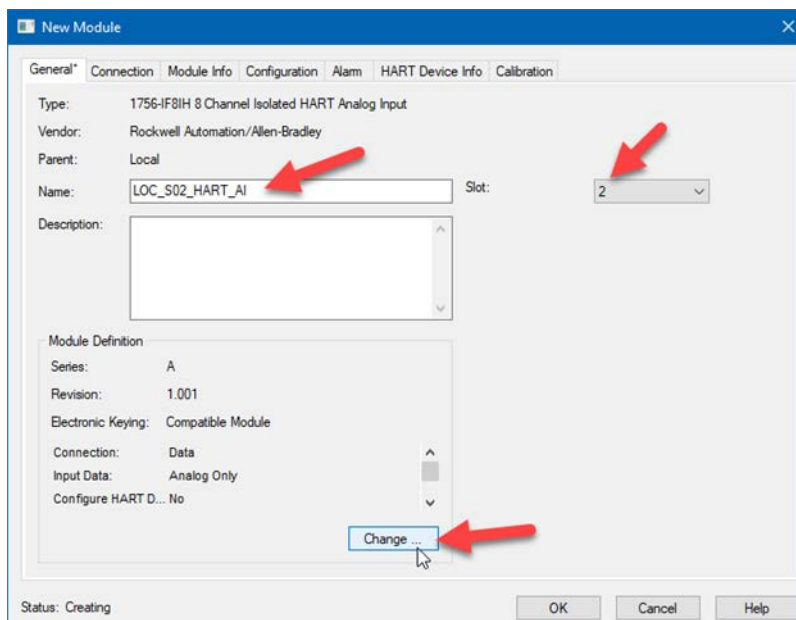
1. In the controller Organizer for your project, select the 1756 Backplane. Right-click and select "New Module...".



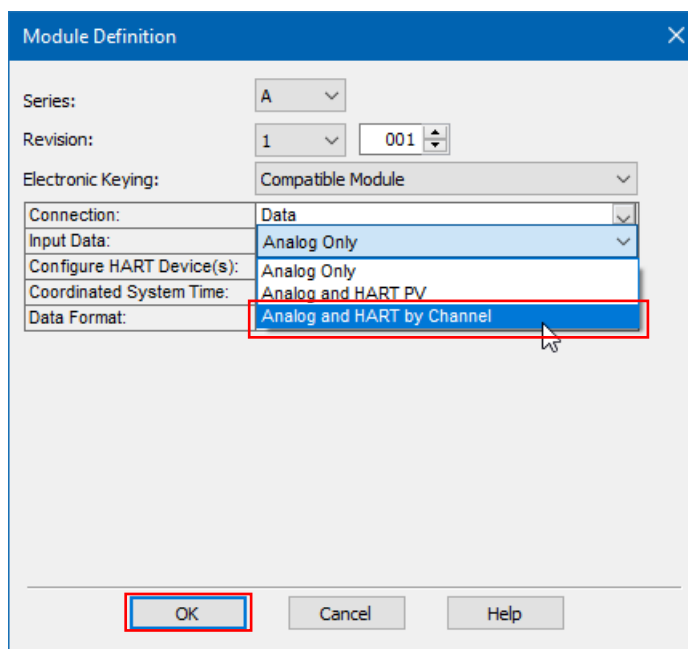
2. Select 1756-IF8IH.



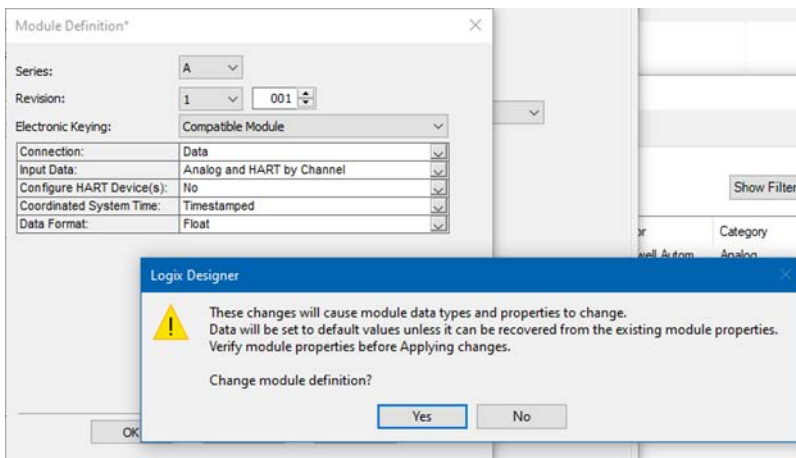
3. Enter a name for the module, Select the slot number where the module is installed, and select Change in the Module Definition.



4. In the Module Definition dialog, change the Input Data selection to "Analog and HART by Channel".

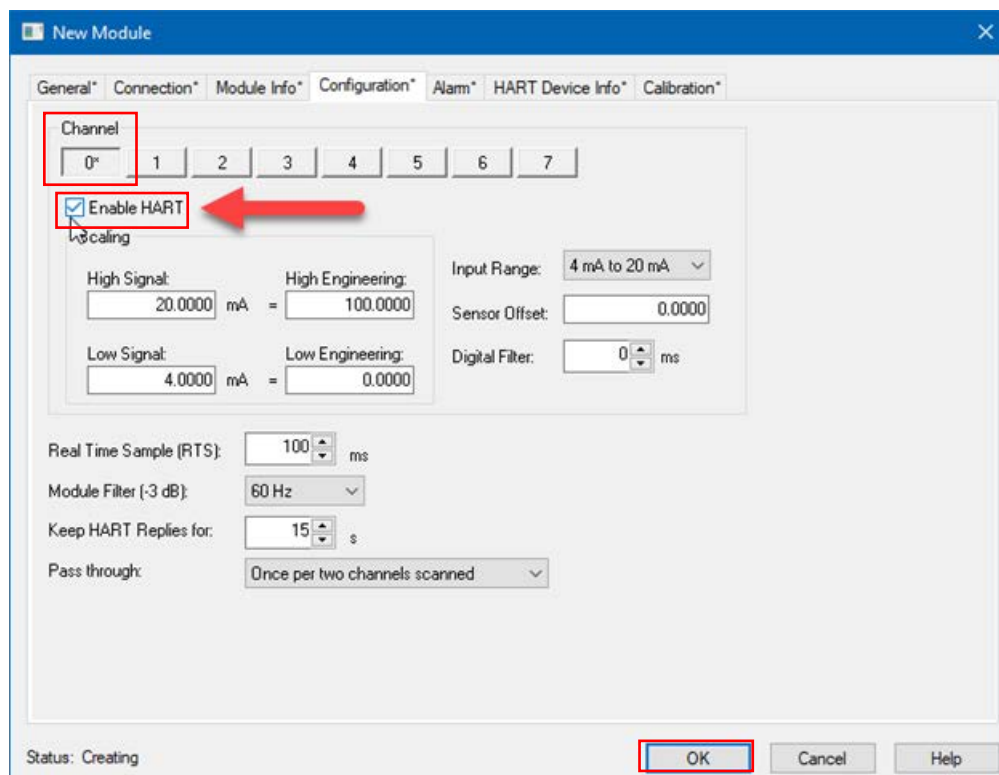


5. Select YES to change the module definition.



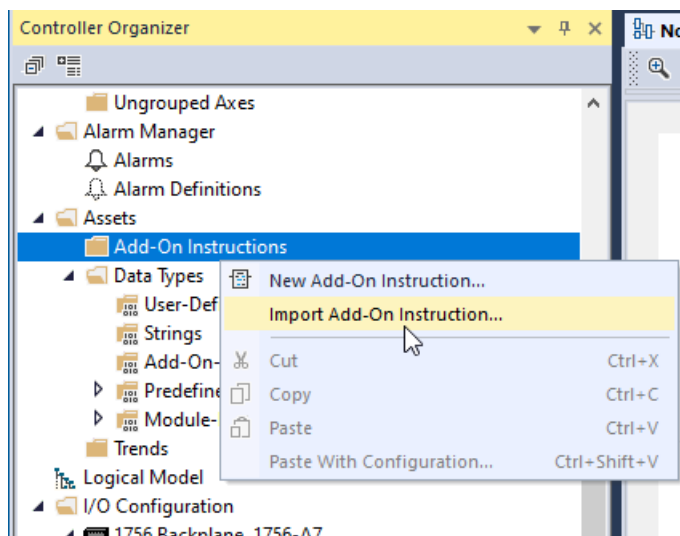
Configure the Channel for the HART Device

1. From the New Module dialog box, Select the configuration tab.
2. Select the Channel where the transmitter is installed and Enable HART on the channel.

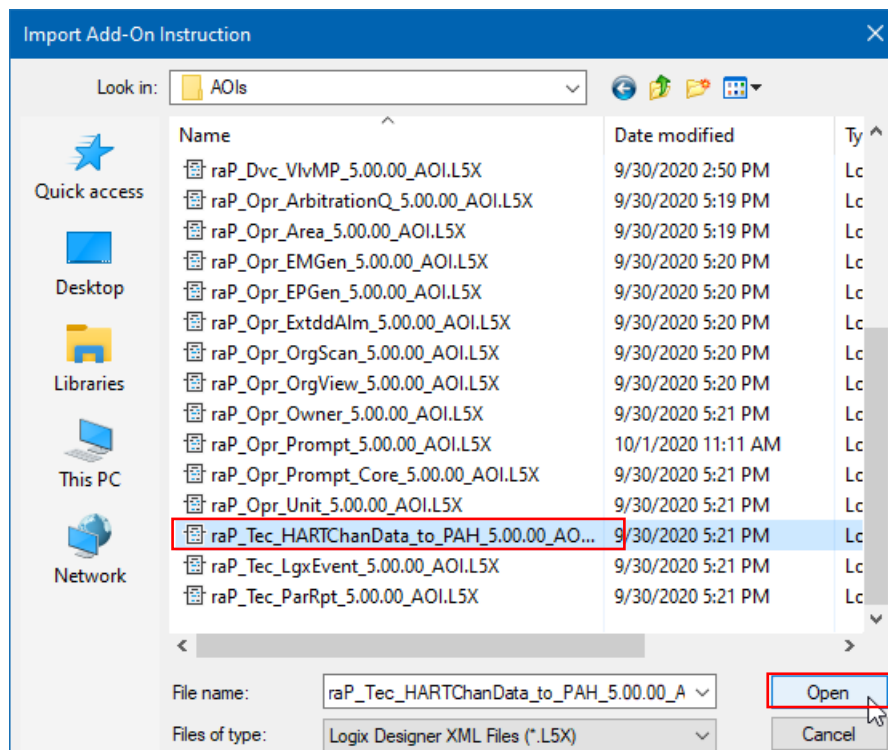


Import the raP_Tec_HARTChanData_to _PAH Add-On Instruction

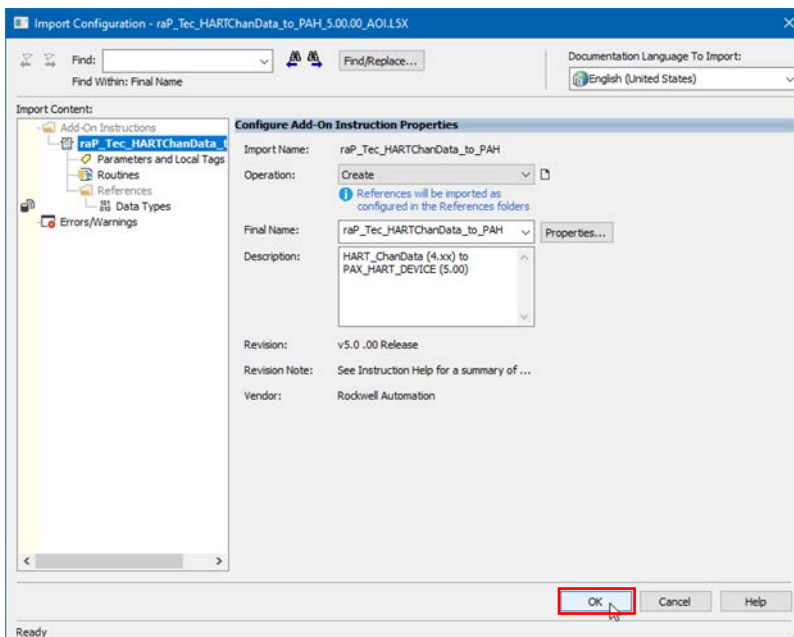
1. In the Controller Organizer, expand "Assets" to show the "Add-On Instructions" folder.
2. Right-click the Add-On Instructions folder and select "Import Add-On Instruction..."



3. Navigate to the location where you downloaded the Library of Process Objects version 5.x.
4. Navigate to the Logix Add-On Instructions. Select the "raP_Tec_HARTChanData_to_PAH" Add-On Instruction import L5X file.

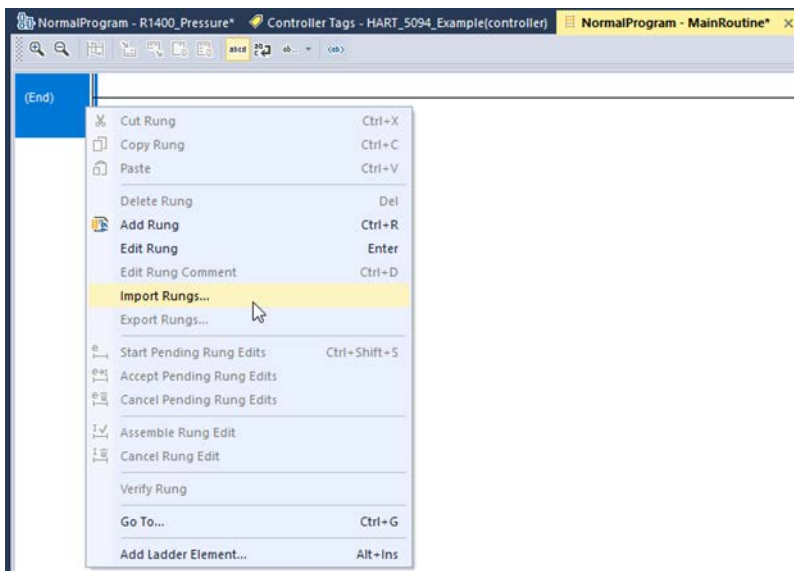


5. Select OK in the Import Configuration dialog box to accept the default values.



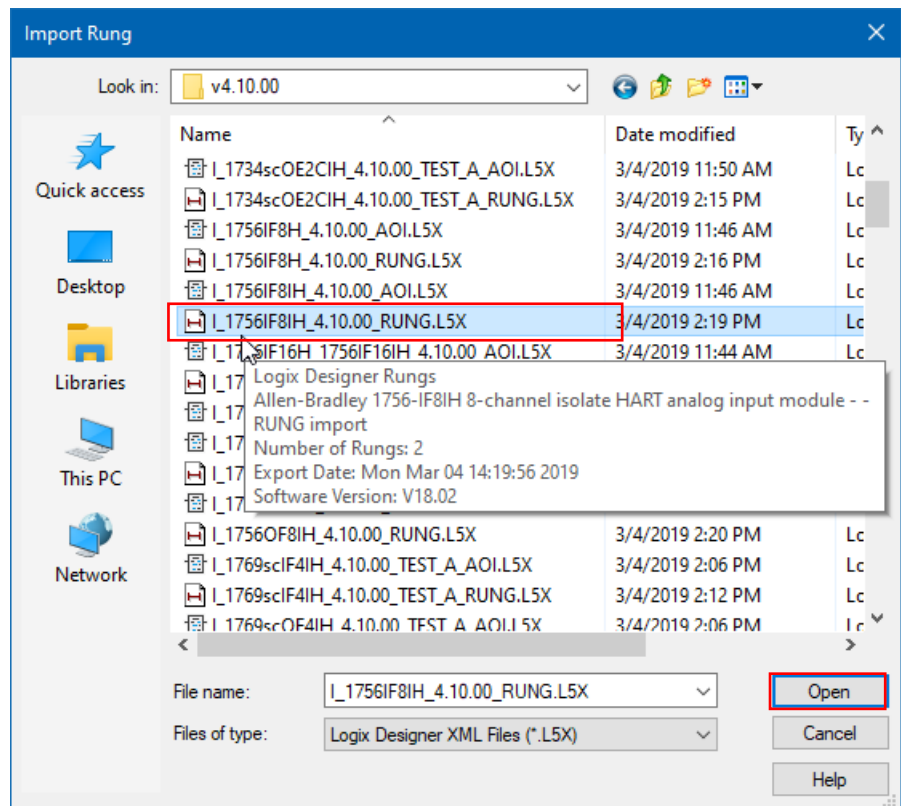
Import the I_1756IF8IH Rung into the Project

1. Open a Ladder Diagram routine in your project.
2. Click in the left margin where you want to insert the rung for the 1756-IF8IH module. Right-click and select "Import Rungs..."

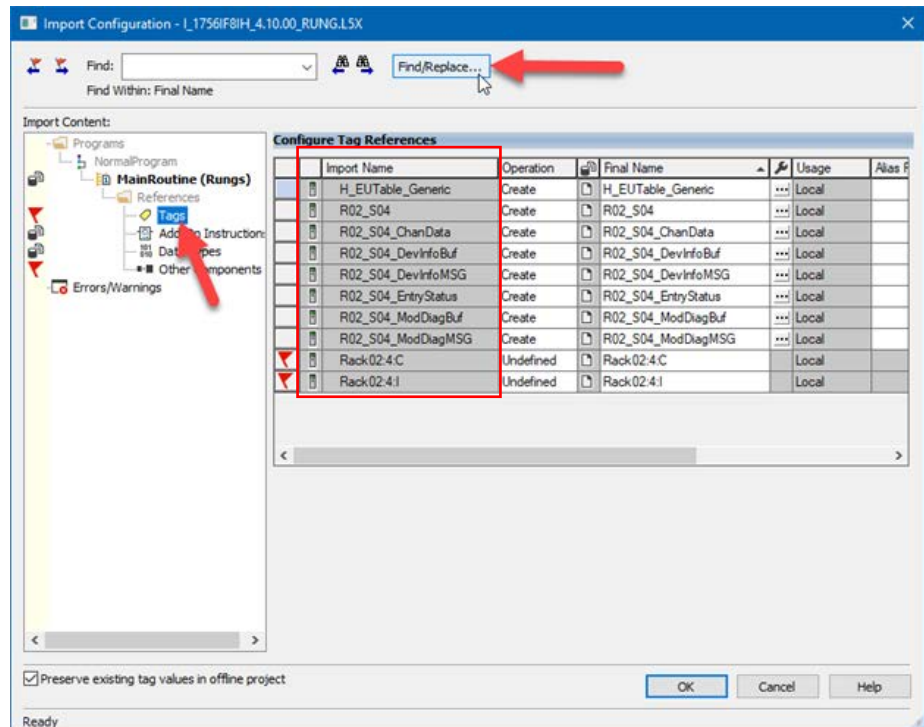


3. Navigate to the location where you downloaded the Library of Process Objects version 4.10.xx.

4. Navigate to the Logix Add-On Instructions. Select the I_1756IF8IH_4.10.00_RUNG.L5X file.

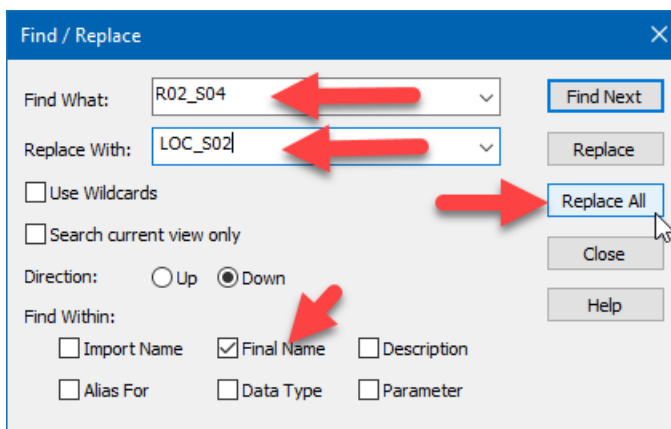


5. In the Import Configuration window, select the "Tags" item in the "Import Content" tree on the left. Note the names of tags in the import file. Select "Find/Replace..."

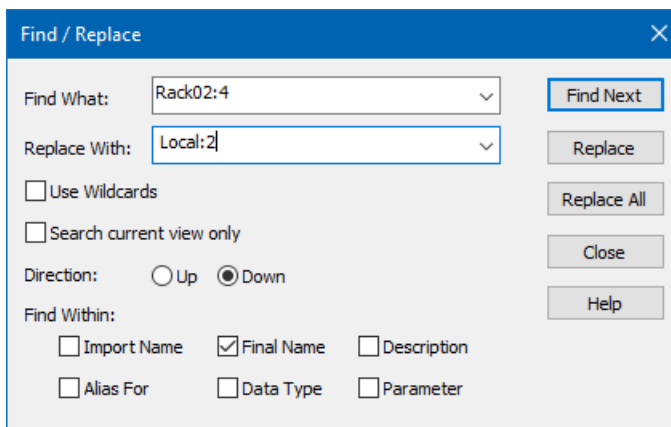


6. Change the base that you want to use for the tag names.
- In the "Find What" box, enter "R02_S04", which is the base name for the tags in the import file.

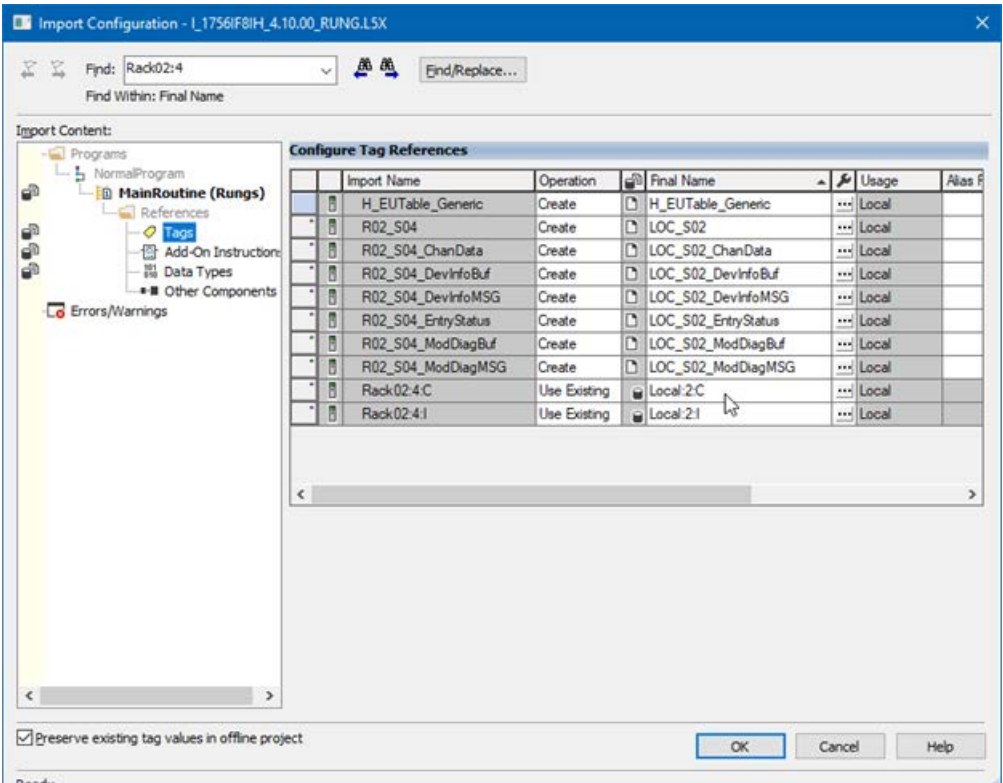
- In the "Replace With" box, enter the base that you want to use for tag names for this rung. Since we created the module in local chassis slot 2, for this example we use "LOC_S02".
- Select "Replace All"



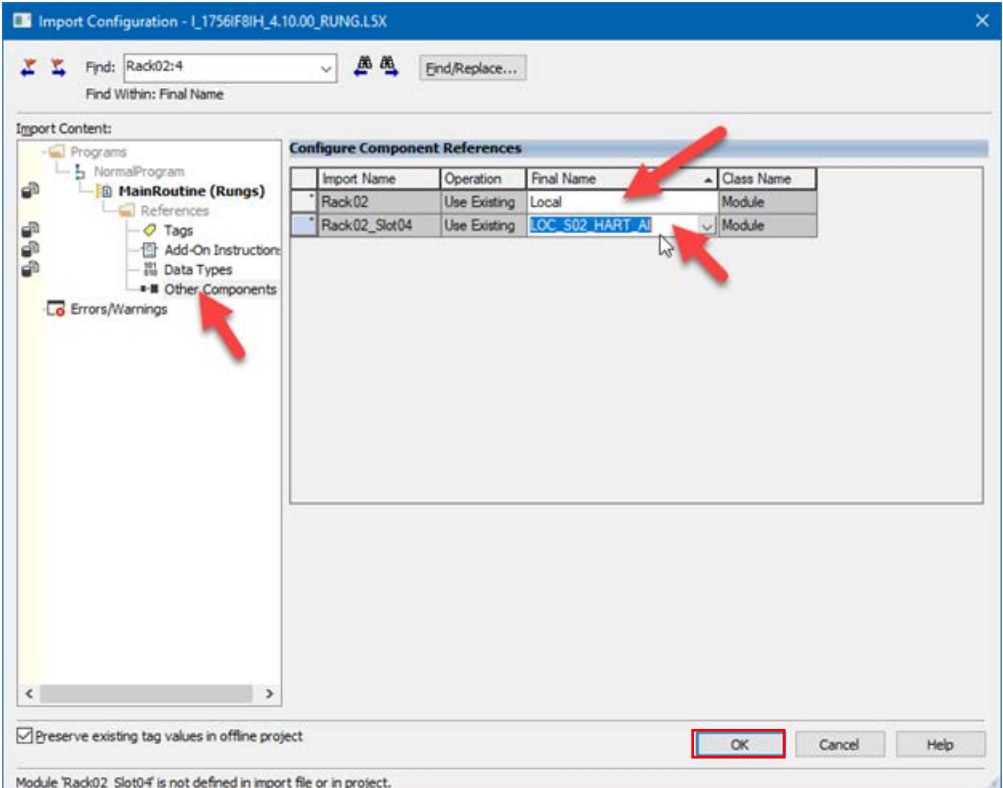
7. Use the same process to replace the text "Rack02:4" in the import with "Local:2", for the tag names assigned to the module I/O data.



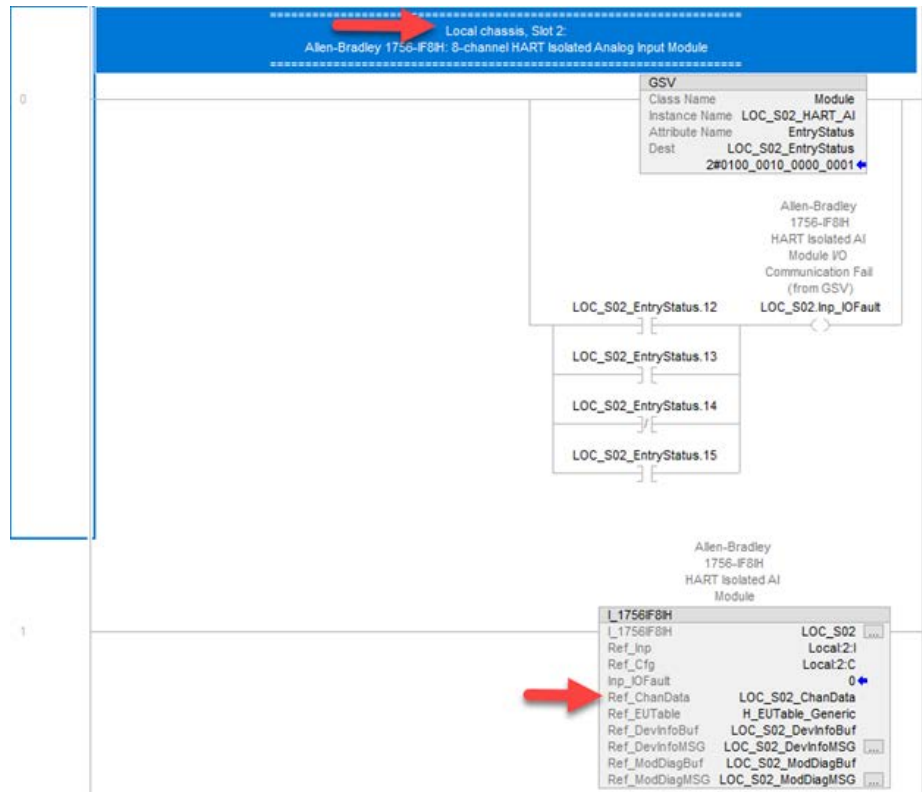
The Final Name column shows the tags to be created or used.



- 8. Select the Other Components item in the Import Contents tree.
- 9. Change the Final Name items to align with the Rack name and the Module name you gave the 1756-IF8IH module when you created it. Select OK to import the rung.



- Two rungs are imported. On the first rung, change the Rung Comment to reflect the location of the module created. Note the tag of the Ref_ChanData InOut parameter in the second rung. This tag name is used in the following steps.

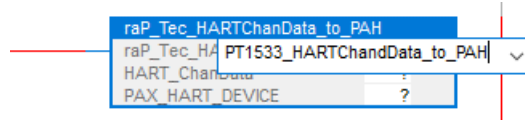


Add the raP_Tec_HARTChanData_to_PAH Instance to the Project

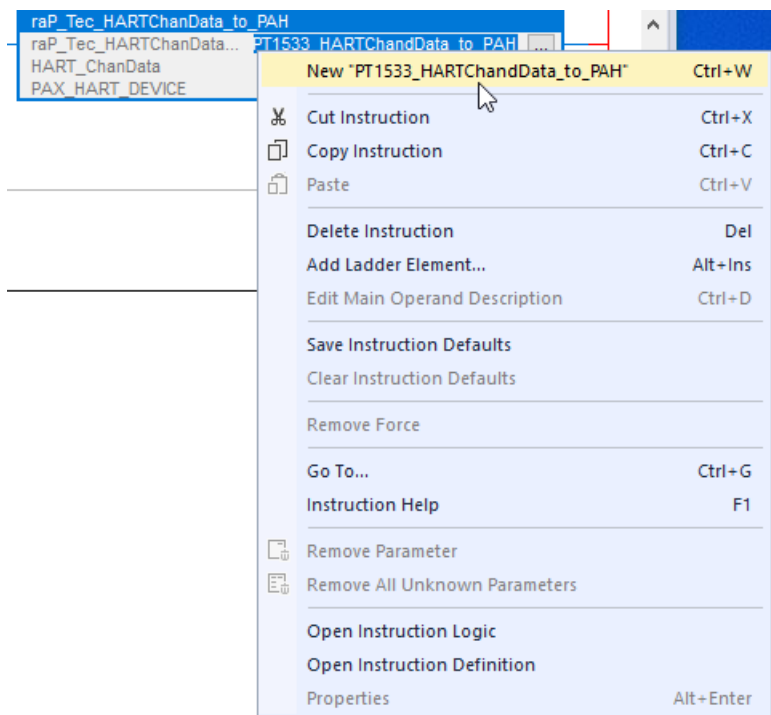
- Add a rung after the L1756IF8IH rung. On that rung, place an instance of the raP_Tec_HARTChanData_to_PAH instruction.



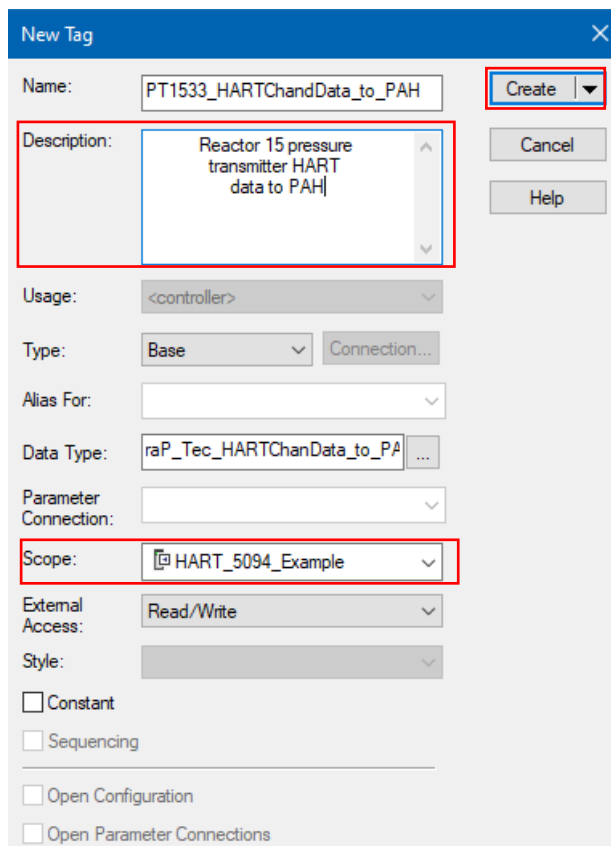
- The first operand is the backing tag for the instruction. Enter a suitable name.



- Right-click and select "New (tag name)".



- Enter a description and select the tag scope. The tag Data Type is set for you automatically.

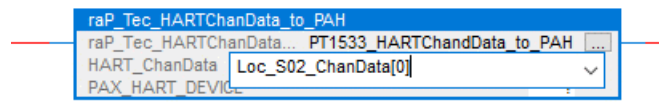


- The second operand is a HART Channel Data member from the I_1756IF8IH instruction.



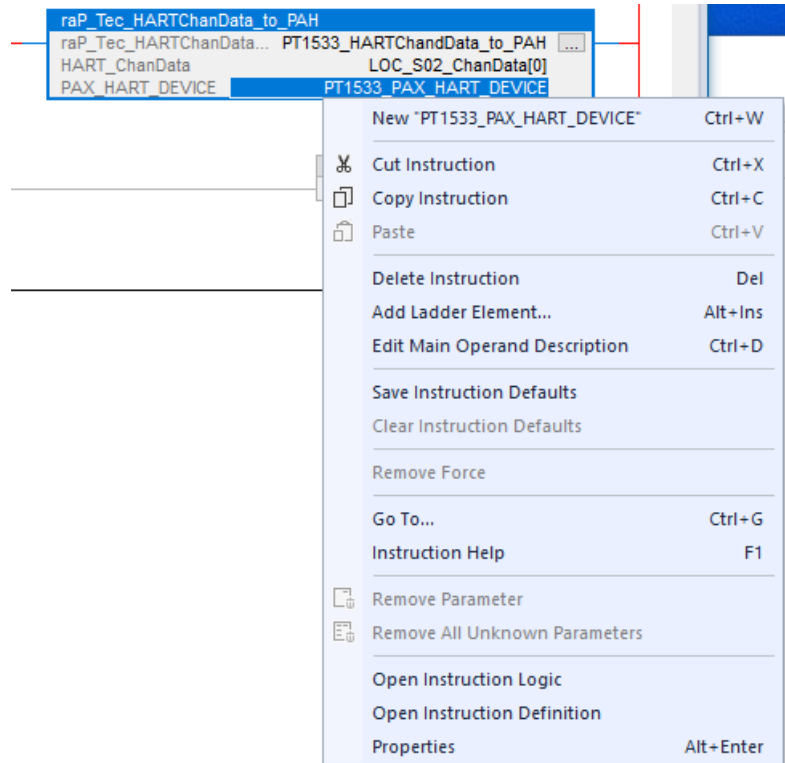
The I_1756IF8IH instruction creates an array of 8 channels' data. Previously we configured Channel 0 on the 1756-IF8IH for this device.

Select element [0] of that array for this operand.



- The third operand is a tag that you create that is the same data type as used by newer HART I/O modules, such as the 5094-IF8IH. This tag contains the HART data coming out of the raP_Tec_HARTChanData_to_PAH instruction and going to the PAH instruction.

Enter a suitable tag name, then right-click and select "New (tag name)".



7. Enter a description and select the tag scope. The tag Data Type is set for you automatically.

New Tag

Name: PT1533_PAX_HART_DEVICE

Description: Reactor 15 pressure transmitter HART data

Usage: <controller>

Type: Base

Alias For:

Data Type: PAX_HART_DEVICE:I:0

Parameter Connection:

Scope: HART_5094_Example

External Access: Read/Write

Style:

☐ Constant

☐ Sequencing

☐ Open Configuration

☐ Open Parameter Connections

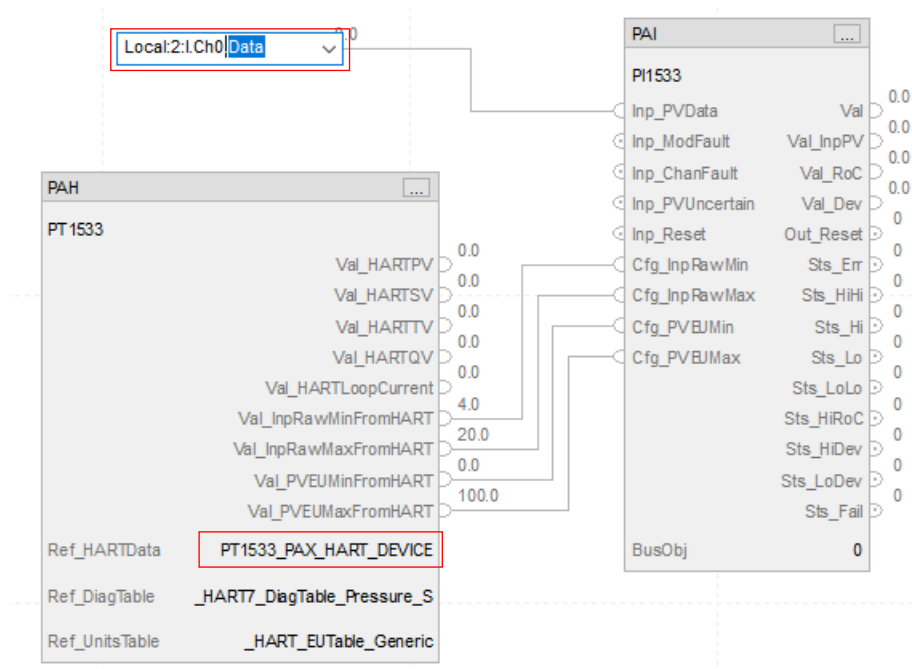
Create Cancel Help

Add the PAH and PAI Instances to the Project and Connect PAH and PAI Instances

Continue as in the example documented in [Appendix A, Add the PAH \(Process Analog HART\) and PAI \(Process Analog Input\) Instruction Instances to the Project](#), creating the PAH and PAI instances and linking them together.

The "Ref_HARTData" operand on the PAH instruction is the tag that you just created above, PT1533_PAX_HART_DEVICE. The analog input to the PAI instruction comes from the input data value from Channel 0 of the 1756-IF8IH, which is in the Local chassis, slot 2. In this example, the tag is Local:2:I.Ch0.Data.

The following diagram shows the final configuration for this example.



Notes:

Rockwell Automation Support

Use these resources to access support information.

Technical Support Center	Find help with how-to videos, FAQs, chat, user forums, Knowledgebase, and product notification updates.	rok.auto/support
Local Technical Support Phone Numbers	Locate the telephone number for your country.	rok.auto/phonesupport
Technical Documentation Center	Quickly access and download technical specifications, installation instructions, and user manuals.	rok.auto/techdocs
Literature Library	Find installation instructions, manuals, brochures, and technical data publications.	rok.auto/literature
Product Compatibility and Download Center (PCDC)	Download firmware, associated files (such as AOP, EDS, and DTM), and access product release notes.	rok.auto/pcdc

Documentation Feedback

Your comments help us serve your documentation needs better. If you have any suggestions on how to improve our content, complete the form at rok.auto/docfeedback.

Waste Electrical and Electronic Equipment (WEEE)



At the end of life, this equipment should be collected separately from any unsorted municipal waste.

Rockwell Automation maintains current product environmental compliance information on its website at rok.auto/pec.

Allen-Bradley, ArmorStart, CompactLogix, ControlLogix, FactoryTalk, FactoryTalk Optix, iTRAK, Kinetix, MagneMotion, PhaseManager, PlantPAx, PowerFlex, Rockwell Automation, RSLogix, RSLogix 5000, SoftLogix, Stratix, Studio 5000, Studio 5000 Logix Designer, and TechConnect are trademarks of Rockwell Automation, Inc.

Excel is a trademark of Microsoft.

EtherNet/IP is a trademark of ODVA, Inc.

Trademarks not belonging to Rockwell Automation are property of their respective companies.

Rockwell Otomasyon Ticaret A.Ş. Kar Plaza İş Merkezi E Blok Kat:6 34752, İçerenköy, İstanbul, Tel: +90 (216) 5698400 EEE Yönetmeliğine Uygundur

Connect with us.    

rockwellautomation.com — expanding **human possibility**®

AMERICAS: Rockwell Automation, 1201 South Second Street, Milwaukee, WI 53204-2496 USA, Tel: (1) 414.382.2000, Fax: (1) 414.382.4444

EUROPE/MIDDLE EAST/AFRICA: Rockwell Automation NV, Pegasus Park, De Kleetlaan 12a, 1831 Diegem, Belgium, Tel: (32) 2663 0600, Fax: (32) 2 663 0640

ASIA PACIFIC: Rockwell Automation SEA Pte Ltd, 2 Corporation Road, #04-05, Main Lobby, Corporation Place, Singapore 618494, Tel: (65) 6510 6608, FAX: (65) 6510 6699

UNITED KINGDOM: Rockwell Automation Ltd., Pitfield, Kiln Farm, Milton Keynes, MK11 3DR, United Kingdom, Tel: (44)(1908) 838-800, Fax: (44)(1908) 261-917

Publication PROCES-RM200K-EN-P - June 2026

Supersedes Publication PROCES-RM200J-EN-P - December 2025

Copyright © 2026 Rockwell Automation, Inc. All rights reserved. Printed in the U.S.A.