



Logix 5000 Controllers Design Considerations

ControlLogix, GuardLogix, CompactLogix,
Compact GuardLogix, SoftLogix



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.

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



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



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



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

	Preface	9
	Overview	9
	Summary of Changes	9
	Additional Resources	10
	 Chapter 1	
5580 Controller and 5380 Controllers	ControlLogix 5580 and GuardLogix 5580 Controllers	11
	CompactLogix 5380 and Compact GuardLogix 5380 Controllers.	12
	Process Controllers	13
	Controller Memory	13
	Data Types	14
	Extended Data Types	15
	Programming Techniques	15
	Data Alignment Rules	15
	Produced and Consumed Data	16
	Connections	16
	 Chapter 2	
5480 Controller	CompactLogix 5480 Controller	17
	Controller Memory	18
	Data Types	18
	Extended Data Types	19
	Programming Techniques	19
	Data Alignment Rules	19
	Produced and Consumed Data	20
	Connections	20
	 Chapter 3	
5570 Controllers and 5370 Controllers	ControlLogix 5570 and GuardLogix 5570 Controllers	21
	CompactLogix 5370 and Compact GuardLogix 5370 Controllers.	22
	Controller Memory	23
	CompactLogix 5370 and Compact GuardLogix 5370 Controllers.	23
	Controller Connections	24
	Determine Total Connection Requirements	25
	System Overhead Percentage	26
	Manage the System Overhead Timeslice Percentage	28
	I/O Processing	29
	Data Types	29
	Programming Techniques	30
	Produced and Consumed Data	30
	Messages	30

Logic Execution**Chapter 4**

Decide When to Use Tasks, Programs, and Routines	31
Specify Task Priorities	32
Manage User Tasks.....	33
Pre-defined Tasks in ControlLogix and CompactLogix Process	
Controllers	33
Considerations that Affect Task Execution.....	34
Configure a Continuous Task.....	35
Configure a Periodic Task	35
Configure an Event Task	35
Guidelines to Configure an Event Task.....	36
Additional Considerations for Periodic and Event Tasks.....	36
Access the Module Object	36
Develop Application Code in Routines	37
Comparison of Programming Languages	38
Programming Methods	38
Inline Duplication.....	38
Indexed Routine.....	39
Buffered Routine.....	39
Controller Prescan of Logic	40
Add-On Instruction Prescan Logic.....	40
Controller Postscan of SFC Logic	41
Add-On Instruction Postscan Logic.....	41
Timer Execution	42
SFC Step Timer Execution.....	42
Edit an SFC Online	43

Chapter 5**Modular Programming
Techniques**

Guidelines for Code Reuse.....	46
Naming Conventions.....	47
Parameter Name Prefixes	49
Guidelines for Subroutines.....	50
Guidelines for User-defined Data Types.....	51
Naming Conventions for User-Defined Data Types.....	51
UDT Member Order	51
.....	53
Guidelines for Add-On Instructions	53
Add-On Instruction Design Concepts	55
Naming Conventions for Add-On Instructions	55
Comparison of Subroutines and Add-On Instructions.....	55
Comparison of Partial Import/Export and	
Add-On Instructions	56
Guidelines for Program Parameters	57
Comparison of Program Parameters and	
Add-On Instructions	58
Compare Controller Organizer and Logical Organizer	58

Structure Logic According to Standards	Chapter 6	
	Physical Model.....	60
	Separate a Process Unit into Equipment Modules and Control Modules	62
	Physical Model Naming Conventions	62
	Procedural Model	64
	Identify Operations and Phases	65
	Procedural Control Modes.....	66
	Procedural Control States.....	66
	Procedural Control Commands	67
	Procedural Model Naming Conventions	68
	State Model.....	69
Produced and Consumed Data	Chapter 7	
	Guidelines for Produced and Consumed Tags	72
	Guidelines for Produced and Consumed Axis.....	73
	Guidelines to Specify an RPI Rate for Produced and Consumed Tags.....	73
	Guidelines to Manage Connections for Produced and Consumed Tags.....	73
	Configure an Event Task Based on a Consumed Tag	74
	Compare Messages and Produced/Consumed Tags	74
Data Structures	Chapter 8	
	Guidelines for Data Types	75
	Arrays.....	76
	Guidelines for Arrays	77
	Indirect Addresses of Arrays.....	77
	Guidelines for Array Indexes	78
	Guidelines for User-defined Data Types (UDT)	79
	Select a Data Type for Bit Tags	80
	Serial Bit Addresses.....	81
	Guidelines for String Data Types	82
	PLC-5/SLC 500 Access of Strings.....	82
	Configure Tags	83
	Guidelines for Base Tags	83
	Create Alias Tags.....	84
	Guidelines for Data Scope	85
	Guidelines for Tag Names	85
	Guidelines for Extended Tag Properties.....	86
	Tag Descriptions.....	86
	Protect Data Access Control at Tag Level	87

Communicate with I/O	Chapter 9	
	Buffer I/O Data.....	89
	Guidelines to Specify an RPI Rate for I/O Modules.....	90
	Communication Formats for I/O Modules.....	91
	Direct Connection.....	91
	Rack-optimized Connection.....	91
	Peer Control.....	92
	Electronic Keying.....	93
	More Information.....	93
	Guidelines to Manage I/O Connections.....	94
	Create Tags for I/O Data.....	96
	Controller Ownership.....	97
	Runtime/Online Addition of Modules.....	98
	Online Addition of Module and Connection Types.....	99
	Design Considerations for Runtime/Online Addition of Modules.....	100
Determine the Appropriate Network	Chapter 10	
	EtherNet/IP Network Topology.....	102
	Guidelines for EtherNet/IP Networks.....	103
	ControlNet Network Topology.....	103
	Guidelines for ControlNet Networks.....	104
	Guidelines for Unscheduled ControlNet Networks.....	105
	Compare Scheduled and Unscheduled ControlNet Communication.....	106
	DeviceNet Network Topology.....	106
Communicate with Other Devices	Guidelines for DeviceNet Networks.....	107
	Chapter 11	
	Cache Messages.....	110
	Message Buffers.....	110
	Outgoing Unconnected Buffers.....	111
	Guidelines for Messages.....	112
	Guidelines to Manage Message Connections.....	112
	Guidelines for Block Transfer Messages.....	113
Alarms	Map Tags.....	113
	Chapter 12	
	Guidelines for Tag-Based Alarms.....	115
	Access Tag-based Alarms.....	116
	Guidelines for Instruction-Based Alarms.....	117
	Configure Logix-based Alarm Instructions.....	118
	Automatic Diagnostics.....	119

	Chapter 13	
Optimize an Application for Use with HMI	Linux-based Software Use of Controller Memory	121
	HMI Implementation Options.....	122
	Guidelines for FactoryTalk View Software	122
	How a Data Server Communicates with the Controllers.....	122
	Compare RSLinx Classic and FactoryTalk Linx Software.....	124
	Guidelines for Linux-based Software	124
	Guidelines to Configure Controller Tags.....	125
	Reference Controller Data from FactoryTalk View Software .	125
	Chapter 14	
Develop Equipment Phases	Guidelines for Equipment Phases.....	127
	Equipment Phase Instructions.....	128
	Chapter 15	
Manage Firmware	Guidelines to Manage Controller Firmware	129
	Compare Firmware Options	130
	Guidelines for the Firmware Supervisor	131
	Access Firmware	132
	Index	133

Notes:

Overview

Throughout this publication, the term “programming software” refers to:

- Studio 5000 Logix Designer® application, version 21 or later
- RSLogix 5000® software, version 16 or later

This publication features these controllers, and where applicable, the controllers are known as:

Controller Family	Includes these controllers
5580 controllers	ControlLogix® 5580 and GuardLogix® 5580 controllers
5380 controllers	CompactLogix™ 5380 and Compact GuardLogix 5380 controllers
5480 controllers	CompactLogix 5480 Controllers
5570 controllers	ControlLogix 5570 and GuardLogix 5570 controllers
5370 controllers	CompactLogix 5370 and Compact GuardLogix 5370 controllers

Summary of Changes

This manual contains new and updated information as indicated in the following table.

Topic	Page
Added ControlLogix 5580 NSE, ControlLogix-XT™ 5580, and ControlLogix 5580 Process controllers.	Throughout
Added CompactLogix 5380 Process controllers	Throughout
Added Pre-defined Tasks in ControlLogix and CompactLogix Process Controllers	33
Updated table: Online Addition of Module and Connection Types	99
Added information on Automatic Diagnostics.	119

Additional Resources

These documents contain additional information about the controllers.

Resource	Description
<ul style="list-style-type: none"> EtherNet/IP™ Network Devices User Manual, publication ENET-UM006 ControlNet® Modules in Logix 5000™ Control Systems User Manual, publication CNET-UM001 DeviceNet® Modules in Logix 5000 Control Systems User Manual, publication DNET-UM004 	Networks
<ul style="list-style-type: none"> System Security Design Guidelines Reference Manual, SECURE-RM001 Configure System Security Features User Manual, SECURE-UM001 CIP Security™ with Rockwell Automation Products Application Technique, SECURE-AT001 	Provides guidance on how to conduct vulnerability assessments, implement Rockwell Automation products in a secure system, harden the control system, manage user access, and dispose of equipment.
<ul style="list-style-type: none"> Replacement Guidelines: Logix 5000 Controllers Reference Manual, publication 1756-RM100 Logix 5000 Common Procedures Programming Manual, publication 1756-PM001 	Logix 5000™ Controllers
<ul style="list-style-type: none"> ControlLogix 5580 Controllers User Manual, publication 1756-UM543 ControlLogix System User Manual, publication 1756-UM001 Motion Configuration and Startup User Manual, publication MOTION-UM001 Motion Coordinate System User Manual, publication MOTION-UM002 	ControlLogix Controllers
<ul style="list-style-type: none"> CompactLogix 5370 Controllers User Manual, publication 1769-UM021 CompactLogix 5380 and Compact GuardLogix 5380 Controllers User Manual, publication 5069-UM001 1768 CompactLogix System User Manual, publication 1768-UM001 1769 CompactLogix System User Manual, publication 1769-UM011 1769 Packaged CompactLogix Controllers Quick Start and User Manual, publication IASIMP-QS010 	CompactLogix Controllers

You can view or download publications at
<http://www.rockwellautomation.com/literature/>.

5580 Controller and 5380 Controllers

This chapter highlights these controllers, and where applicable, the controllers are known as:

Controller Family	Includes these controllers
5580 controllers	ControlLogix® 5580 and GuardLogix® 5580 controllers
5380 controllers	CompactLogix™ 5380 and Compact GuardLogix 5380 controllers

ControlLogix 5580 and GuardLogix 5580 Controllers

Characteristic	ControlLogix 5580 Controllers and GuardLogix 5580 Controllers	
Controller tasks:	<ul style="list-style-type: none"> • 32 • 1000 programs/task 	
Event tasks	Consumed tag, EVENT instruction triggers, Module Input Data changes, and motion events	
User memory	1756-L81E, 1756-L81EK, 1756-L81E-NSE, 1756-L81EXT, 1756-L81EP	3 MB
	1756-L82E, 1756-L82EK, 1756-L82E-NSE, 1756-L82EXT	5 MB
	1756-L83E, 1756-L83EK, 1756-L83E-NSE, 1756-L83EXT, 1756-L83EP	10 MB
	1756-L84E, 1756-L84EK, 1756-L84E-NSE, 1756-L84EXT	20 MB
	1756-L85E, 1756-L85EK, 1756-L85E-NSE, 1756-L85EXT, 1756-L85EP	40 MB
	1756-L81ES	3 MB + 1.5 MB safety
	1756-L82ES	5 MB + 2.5 MB safety
	1756-L83ES	10 MB + 5 MB safety
	1756-L84ES	20 MB + 6 MB safety
Built-in ports	Single-port Ethernet port, 10 Mbps/100 Mbps/1 Gbps 1-port USB client	
Communication options	<ul style="list-style-type: none"> • EtherNet/IP™ • ControlNet® • DeviceNet® • Data Highway Plus™ • Remote I/O • SynchLink™ • USB Client 	

Characteristic	ControlLogix 5580 Controllers and GuardLogix 5580 Controllers
Network nodes	Studio 5000 Logix Designer® application, version 30 or later
	1756-L81E, 1756-L81EK, 1756-L81E-NSE, 1756-L81EXT, 1756-L81EP, 1756-L81ES 100
	1756-L82E, 1756-L82EK, 1756-L82E-NSE, 1756-L82EXT, 1756-L82ES 175
	1756-L83E, 1756-L83EK, 1756-L83E-NSE, 1756-L83EXT, 1756-L83EP, 1756-L83ES, 1756-L84E, 1756-L84EK, 1756-L84E-NSE, 1756-L84EXT, 1756-L84ES 250
	1756-L85E, 1756-L85EK, 1756-L85E-NSE, 1756-L85EXT, 1756-L85EP 300
Controller redundancy	Fully supported with Studio 5000 Logix Designer Application version 33 later for ControlLogix 5580 controllers. Uses the same firmware revision as standard ControlLogix 5580 controllers, but requires that redundancy is enabled on the Redundancy tab of the Controller Properties dialog.
Integrated motion	EtherNet/IP

CompactLogix 5380 and Compact GuardLogix 5380 Controllers

Characteristic	CompactLogix 5380 Controllers and Compact GuardLogix 5380 Controllers
Controller tasks: <ul style="list-style-type: none"> Continuous Periodic Event 	<ul style="list-style-type: none"> 32 1000 programs/task
Event tasks	Consumed tag, EVENT instruction triggers, Module Input Data changes, and motion events
User memory	5069-L306ER, 5069-L306ERM 0.6 MB
	5069-L310ER, 5069-L310ER-NSE, 5069-L310ERM 1 MB
	5069-L320ER, 5069-L320ERM, 5069-L320ERMK, 5069-L320ERP 2 MB
	5069-L330ER, 5069-L330ERM, 5069-L330ERMK 3 MB
	5069-L340ER, 5069-L340ERM, 5069-L340ERP 4 MB
	5069-L350ERM, 5069-L320ERMK 5 MB
	5069-L380ERM 8 MB
	5069-L3100ERM 10 MB
	5069-L306ERS2, 5069-L306ERMS2 0.6 MB + 0.3 MB safety
	5069-L310ERS2, 5069-L310ERMS2 1 MB + 0.5 MB safety
	5069-L320ERS2, 5069-L320ERMS2, 5069-L320ERS2K, 5069-L320ERMS2K 2 MB + 1 MB safety
	5069-L330ERS2, 5069-L330ERMS2, 5069-L330ERS2K, 5069-L330ERMS2K 3 MB + 1.5 MB safety
	5069-L340ERS2, 5069-L340ERMS2 4 MB + 2 MB safety
	5069-L350ERS2, 5069-L350ERMS2, 5069-L350ERS2K, 5069-L350ERMS2K 5 MB + 2.5 MB safety
	5069-L380ERS2, 5069-L380ERMS2 8 MB + 4 MB safety
	5069-L3100ERS2, 5069-L3100ERMS2 10 MB + 5 MB safety
<ul style="list-style-type: none"> Built-in ports 	<ul style="list-style-type: none"> 2 - Ethernet ports, 10 Mbps/100 Mbps/1 Gbps 1-port USB client
<ul style="list-style-type: none"> Communication options 	<ul style="list-style-type: none"> EtherNet/IP USB Client

Characteristic	CompactLogix 5380 Controllers and Compact GuardLogix 5380 Controllers	
Network nodes	Studio 5000 Logix Designer application, version 31 or later	
	5069-L306ER, 5069-L306ERM, 5069-L306ERS2, 5069-L306ERMS2	16
	5069-L310ER, 5069-L310ER-NSE, 5069-L310ERM, 5069-L310ERS2, 5069-L310ERMS2	24
	5069-L320ER, 5069-L320ERM, 5069-L320ERMK, 5069-L320ERP, 5069-L320ERS2, 5069-L320ERMS2, 5069-L320ERS2K, 5069-L320ERMS2K	40
	5069-L330ER, 5069-L330ERM, 5069-L330ERMK, 5069-L330ERS2, 5069-L330ERMS2, 5069-L330ERS2K, 5069-L330ERMS2K	60
	5069-L340ER, 5069-L340ERM, 5069-L340ERP, 5069-L340ERS2, 5069-L340ERMS2	90
	5069-L350ERM, 5069-L350ERMK, 5069-L350ERS2, 5069-L350ERMS2, 5069-L350ERS2K, 5069-L350ERMS2K	120
	5069-L380ERM, 5069-L380ERS2, 5069-L380ERMS2	150
	5069-L3100ERM, 5069-L3100ERS2, 5069-L3100ERMS2	180
Controller redundancy	Logix Hot Backup - CompactLogix 5380 Controllers only	
Integrated motion	EtherNet/IP	

Process Controllers

ControlLogix 5580 and CompactLogix 5380 process controllers are extensions of the Logix 5000™ controller family that focus on plant-wide process control.

The process controllers come configured with a default process tasking model and dedicated PlantPAx® process instructions that are optimized for process applications and that improve design and deployment efforts. The process controllers support release 5.0 of the Rockwell Automation Library of Process Objects.

For more information on the process library, see the Rockwell Automation Library of Process Objects Reference Manual, publication [PROCES-RM200](#).

For more information on process controller application guidelines, see the PlantPAx DCS Configuration and Implementation User Manual, publication [PROCES-UM100](#).

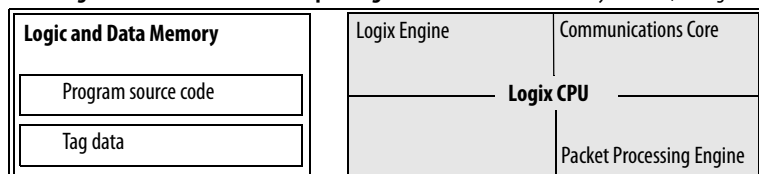
Controller Memory

The Logix CPU runs control and motion, communications, and packet processing each on a separate core.

- The Logix Engine executes the user program, the control task, and the motion task.
- The Communications core manages all Class 3 and unconnected communications via the Ethernet, USB, and backplane communication ports. Communications do not interrupt the user task. The System Overhead Time Slice Percentage setting is no longer available and not necessary.
- The Packet Processing Engine moves all Ethernet Class 1 packets to and from the wire, and moves all packets to and from the backplane.

The controller allocates memory as needed to help prevent many runtime errors that are related to free memory. Runtime memory no longer consumes application memory space.

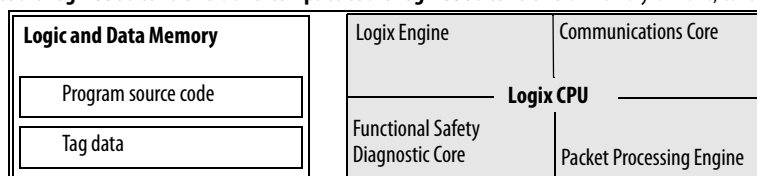
1756 ControlLogix 5580 controllers and CompactLogix 5380 controllers- Memory is in one, contiguous section.



The GuardLogix CPU performs the same functions as the ControlLogix 5580 and CompactLogix 5380 controllers, with these differences:

- The Logix Engine executes the user program, the control task, the motion task, and the safety task.
- The Functional Safety Diagnostic Core runs the safety task with inverted data, and compares the results to the safety task that runs on the Logix Engine.

1756 GuardLogix 5580 controllers and Compact GuardLogix 5380 controllers- Memory is in one, contiguous section.



Data Types

The controllers support IEC 61131-3 atomic data types. The controllers also support compound data types, such as arrays, predefined structures (such as counters and timers), and user-defined structures).

The Logix CPU reads and manipulates 32-bit data values. The minimum memory allocation for data in a tag is 4 bytes. When you create a standalone tag that stores data that is less than 4 bytes, the controller allocates 4 bytes, but the data only fills the part that it needs.

For more information See [Data Structures on page 75](#).

Data Type	Bits												
	64...32	31	16	15	8	7	1	0					
BOOL	Not allocated	Allocated but not used							0 or 1				
SINT	Not allocated	Allocated but not used					-128...+127						
INT	Not allocated	Allocated but not used			-32,768...32,767								
DINT	Not allocated	-2,147,483,648...2,147,483,647											
REAL	Not allocated	-3.40282347E ³⁸ ...-1.17549435E ⁻³⁸ (negative values) 0 1.17549435E ⁻³⁸ ...3.40282347E ³⁸ (positive values)											
LINT		-922337203685477580...+9223372036854775807											

Extended Data Types

The 5380 and 5580 controllers support these extended data types:

Data Type	Bits						
	64...32	31	16	15	8	7...1	0
USINT	Not allocated	Allocated but not used				Unsigned 0...255	
UINT	Not allocated	Allocated but not used			Unsigned 0...65,535		
UDINT	Not allocated	Unsigned 0...4,294,967,295					
ULINT	Unsigned 0...18,446,744,073,709,551,615						
LREAL	-1.7976931348623157E308...-2.2250738585072014E-308 (negative values) 0.0 2.2250738585072014E-308...1.7976931348623157E308 (positive values)						

The compute, compare, and math instructions support these extended data types for 64-bit operations.

Programming Techniques

Programming Technique	Consideration
Subroutines	For Logix Designer application Version 28 and later on 5580 and 5380 controllers: <ul style="list-style-type: none"> JSR calls are limited to 40 input parameters and 40 output parameters. There is a maximum of 25 JSR nesting levels.
Add-On Instructions	For 5580 controllers 5380 controllers, you can nest Add-On Instructions up to 25 levels.
PhaseManager™ equipment phases	The PhaseManager option is support on 5580 and 5380 controllers as of firmware revision 32.

For more information See [Modular Programming Techniques on page 45](#).

Data Alignment Rules

The 5580, 5380, and all 64-bit controllers have these data alignment rules on UDTs:

- 8-byte (64-bit) data types (LINT, ULINT, and LREAL) are placed on 8-byte address boundaries in RAM. The Studio 5000 Logix Designer application manages this requirement automatically.
- UDTs that have no 8-byte elements retain the existing 4-byte memory allocation rules.
- UDTs that contain LINTs are considered to be 8-byte data types and their size is a multiple of 8 bytes.
- 8-byte data types (LINTs or embedded UDTs) within a data structure are aligned on an 8-byte boundary.

Produced and Consumed Data

The controller supports:

- Total number of produced tags ≤ 255
- Maximum number of multicast produce tags out of the Ethernet port ≤ 32
- Maximum number of consumed tags ≤ 255

For more information See [Produced and Consumed Data on page 71](#).

Connections

The controller supports:

- Dedicated Class 1 (I/O, Produce and Consume, implicit, and so on) connection pool to support controller node count
- Dedicated Class 3 (HMI, message instructions, explicit, and so on) connection pool to support up to 512 connections
 - This pool is split; 256 incoming and 256 outgoing connections
- 256 cached buffers
- 320 unconnected buffers for establishing connections
 - This value is fixed and cannot be increased with a CIP™ Generic message instruction.

5480 Controller

CompactLogix 5480 Controller

Characteristic	CompactLogix™ 5480 Controller	
Controller tasks: <ul style="list-style-type: none"> Continuous Periodic Event 	<ul style="list-style-type: none"> 32 tasks 1000 programs/task All event triggers 	
Event tasks	Consumed tag, EVENT instruction triggers, Module Input Data changes, and motion events	
User memory	Windows 10 (commercial operating system on controller)	<ul style="list-style-type: none"> RAM: 6 GB SSD: 64 GB
	Logix control engine	
	5069-L430ERMW	3 MB
	5069-L450ERMW	5 MB
	5069-L4100ERMW	10 MB
	5069-L4200ERMW	20 MB
Built-in ports	<p>Logix control engine use:</p> <ul style="list-style-type: none"> 3 - Ethernet, 10 Mbps/100 Mbps/1 Gbps 1 - USB client <p>IMPORTANT: Consider the following</p> <ul style="list-style-type: none"> When the controller operates in Dual-IP mode, each Ethernet port requires a unique IP address. When the controller operates in Linear/DLR mode, the controller uses only one IP address. <p>Windows 10 use:</p> <ul style="list-style-type: none"> 1 - Ethernet, 10 Mbps/100 Mbps/1 Gbps 2 - USB 3.0 ports 1 - DisplayPort 	
Communication options	<ul style="list-style-type: none"> Dual-port EtherNet/IP™ USB Client 	
Network nodes	Studio 5000 Logix Designer® application, version 32.00.00 or later	
	5069-L430ERMW	60
	5069-L450ERMW	120
	5069-L4100ERMW	180
	5069-L4200ERMW	250
	5069-L46ERMW	250
Controller redundancy	None	
Integrated motion	Total axis count	512 (Any combination of physical, virtual, or consumed axes)
	Virtual axis, max	512
	Position-loop axis, max	150
	Axes/ms, max	100

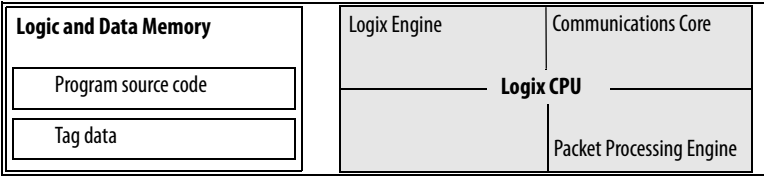
Controller Memory

The Logix CPU runs control and motion, communications, and packet processing each on a separate core.

- The Logix Engine executes the user program, the control task, and the motion task.
- The Communications core manages all Class 3 and unconnected communications via the Ethernet, USB, and backplane communication ports. Communications do not interrupt the user task, and you do not need to adjust the System Overhead Time Slice Percentage.
- The Packet Processing Engine moves all Ethernet Class 1 packets to and from the wire, and moves all packets to and from the backplane.

The controller allocates memory as needed to help prevent many runtime errors that are related to free memory. Runtime memory no longer consumes application memory space.

1756 CompactLogix 5480 controllers- Memory is in one, contiguous section.



Data Types

The controllers support IEC 61131-3 atomic data types. The controllers also support compound data types, such as arrays, predefined structures (such as counters and timers, and user-defined structures.)

The Logix CPU reads and manipulates 32-bit data values. The minimum memory allocation for data in a tag is 4 bytes. When you create a standalone tag that stores data that is less than 4 bytes, the controller allocates 4 bytes, but the data only fills the part that it needs.

For more information See [Data Structures on page 75](#).

Data Type	Bits							
	64...32	31	16	15	8	7	1	0
BOOL	Not allocated	Allocated but not used						0 or 1
SINT	Not allocated	Allocated but not used					-128...+127	
INT	Not allocated	Allocated but not used			-32,768...32,767			
DINT	Not allocated	-2,147,483,648...2,147,483,647						
REAL	Not allocated	-3.40282347E ³⁸ ...-1.17549435E ⁻³⁸ (negative values) 0 1.17549435E ⁻³⁸ ...3.40282347E ³⁸ (positive values)						
LINT	-9223372036854775808...+9223372036854775807							

Extended Data Types

The 5480 controller supports these extended data types:

Data Type	Bits						
	64...32	31	16	15	8	7...1	0
USINT	Not allocated	Allocated but not used				Unsigned 0...255	
UINT	Not allocated	Allocated but not used			Unsigned 0...65,535		
UDINT	Not allocated	Unsigned 0...4,294,967,295					
ULINT	Unsigned 0...18,446,744,073,709,551,615						
LREAL	-1.7976931348623157E308...-2.2250738585072014E-308 (negative values) 0.0 2.2250738585072014E-308...1.7976931348623157E308 (positive values)						

The compute, compare, and math instructions support these extended data types for 64-bit operations.

Programming Techniques

Programming Technique	Consideration
Subroutines	For Logix Designer application Version 32.00.00 and later: <ul style="list-style-type: none"> JSR calls are limited to 40 input parameters and 40 output parameters. There is a maximum of 25 JSR nesting levels.
Add-On Instructions	You can nest Add-On Instructions up to 25 levels.
PhaseManager™ equipment phases	The PhaseManager option is supported on 5480 controllers as of firmware revision 32.

For more information See [Modular Programming Techniques on page 45](#).

Data Alignment Rules

The 5480 controllers have these data alignment rules on UDTs:

- 8-byte (64-bit) data types (LINT, ULINT, and LREAL) are placed on 8-byte address boundaries in RAM. The Studio 5000 Logix Designer application manages this requirement automatically.
- UDTs that have no 8-byte elements retain the existing 4-byte memory allocation rules.
- UDTs that contain LINTs are considered to be 8-byte data types and their size is a multiple of 8 bytes.
- 8-byte data types (LINTs or embedded UDTs) within a data structure are aligned on an 8-byte boundary.

Produced and Consumed Data

The controller supports:

- Total number of produced tags ≤ 255
- Maximum number of multicast produce tags out of the Ethernet port ≤ 32
- Maximum number of consumed tags ≤ 255

For more information See [Produced and Consumed Data on page 71](#).

Connections

The controller supports:

- Dedicated Class 1 (I/O, Produce and Consume, implicit, and so on) connection pool to support controller node count
- Dedicated Class 3 (HMI, message instructions, explicit, and so on) connection pool to support up to 512 connections
 - This pool is split; 256 incoming and 256 outgoing connections
- 256 cached buffers
- 320 unconnected buffers for establishing connections
 - This value is fixed and cannot be increased with a CIP™ Generic message instruction.

5570 Controllers and 5370 Controllers

This chapter highlights these controllers, and where applicable, the controllers are known as:

Controller Family	Includes these controllers
5570 controllers	ControlLogix® 5570 and GuardLogix® 5570 controllers
5370 controllers	CompactLogix™ 5370 and Compact GuardLogix 5370 controllers

ControlLogix 5570 and GuardLogix 5570 Controllers

Characteristic	ControlLogix 5570 Controllers GuardLogix 5570 Controllers Armor™ ControlLogix 5570 Controllers Armor™ GuardLogix® 5570 Controllers
Controller tasks: <ul style="list-style-type: none"> Continuous Periodic Event 	<ul style="list-style-type: none"> 32 1000 programs/task
Event tasks	Consumed tag, EVENT instruction triggers, Module Input Data changes, and motion events
User memory	1756-L71, 1756-L71EROM 2 MB
	1756-L72, 1756-L72EROM 4 MB
	1756-L73, 1756-L73XT, 1756-L73EROM 8 MB
	1756-L74 16 MB
	1756-L75 32 MB
	1756-L71S, 1756-L71EROMS 2 MB + 1 MB safety
	1756-L72S, 1756-L72EROMS 4 MB + 2 MB safety
	1756-L73S, 1756-L73EROMS 8 MB + 4 MB safety
Built-in ports	1756-L71, 1756-L72, 1756-L73, 1756-L73XT, 1756-L74, 1756-L75, 1756-L71S, 1756-L72S, 1756-L73S 1-port USB Client
	1756-L71EROM, 1756-L71EROMS, 1756-L72EROM, 1756-L72EROMS, 1756-L73EROM, 1756-L73EROMS <ul style="list-style-type: none"> 1-port USB client Dual-port EtherNet/IP™, 10 Mbps/100 Mbps
Communication options	<ul style="list-style-type: none"> EtherNet/IP ControlNet® DeviceNet® Data Highway Plus™ Remote I/O SynchLink™ USB Client
Controller connections	500 connections
Controller redundancy	1756-L71, 1756-L72, 1756-L73, 1756-L73XT, 1756-L74, and 1756-L75 controllers only. Full support with a separate redundancy firmware revision.
Integrated motion	EtherNet/IP

CompactLogix 5370 and Compact GuardLogix 5370 Controllers

Characteristic	CompactLogix 5370 Controllers and Compact GuardLogix 5370 Controllers Armor CompactLogix 5370 Controllers and Armor Compact GuardLogix 5370 Controllers	
Controller tasks: <ul style="list-style-type: none"> Continuous Periodic Event 	<ul style="list-style-type: none"> 32 1000 programs/task 	
Event tasks	Consumed tag, EVENT instruction triggers, and motion events	
User memory	1769-L16ER-BB1B	384 KB
	1769-L18ER-BB1B, 1769-L18ERM-BB1B, 1769-L18ERM-BB1BK	512 KB
	1769-L24ER-QB1B, 1769-L24ER-QBFC1B, 1769-L24ER-QBFC1BK	750 KB
	1769-L19ER-BB1B, 1769-L19ER-BB1BK, 1769-L27ERM-QBFC1B, 1769-L30ER, 1769-L30ERK, 1769-L30ER-NSE, 1769-L30ERM, 1769-L30ERMK	1 MB
	1769-L33ER, 1769-L33ERK, 1769-L33ERM, 1769-L33ERMK, 1769-L33ERMO	2 MB
	1769-L36ERM, 1769-L36ERMO	3 MB
	1769-L37ERM, 1769-L37ERMK, 1769-L37ERMO	4 MB
	1769-L38ERM, 1769-L38ERMK, 1769-L38ERMO	5 MB
	1769-L30ERMS	1 MB + 0.5 MB safety
	1769-L33ERMS, 1769-L33ERMSK, 1769-L33ERMOS	2 MB + 1 MB safety
	1769-L36ERMS, 1769-L36ERMOS	3 MB + 1.5 MB safety
	1769-L37ERMS, 1769-L37ERMSK, 1769-L37ERMOS	4 MB + 1.5 MB safety
	1769-L38ERMS, 1769-L38ERMSK, 1769-L38ERMOS	5 MB + 1.5 MB safety
Built-in ports	Dual-port EtherNet/IP 1-port USB Client	
Communication options	EtherNet/IP Embedded switch Single IP address DeviceNet USB Client	
Controller connections	256 connections	
Network nodes	1769-L16ER-BB1B	4
	1769-L18ER-BB1B, 1769-L18ERM-BB1B, 1769-L18ERM-BB1BK, 1769-L19ER-BB1B, 1769-L19ER-BB1BK	8
	1769-L27ERM-QBFC1B, 1769-L30ER, 1769-L30ERK, 1769-L30ER-NSE, 1769-L30ERM, 1769-L30ERMK, 1769-L30ERMS	16
	1769-L33ER, 1769-L33ERK, 1769-L33ERM, 1769-L33ERMK, 1769-L33ERMS, 1769-L33ERMSK, 1769-L33ERMO, 1769-L33ERMOS	32
	1769-L36ERM, 1769-L36ERMS, 1769-L36ERMO, 1769-L36ERMOS	48
	1769-L37ERM, 1769-L37ERMS, 1769-L37ERMO, 1769-L37ERMOS, 1769-L37ERMK, 1769-L37ERMSK	64
	1769-L38ERM, 1769-L38ERMS, 1769-L38ERMO, 1769-L38ERMOS, 1769-L38ERMK, 1769-L38ERMSK	80

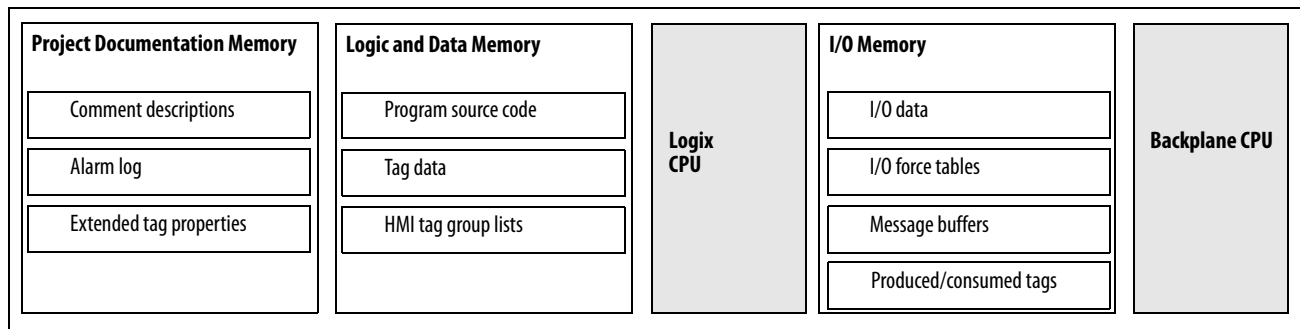
Characteristic	CompactLogix 5370 Controllers and Compact GuardLogix 5370 Controllers Armor CompactLogix 5370 Controllers and Armor Compact GuardLogix 5370 Controllers
Controller redundancy	Back up via DeviceNet - CompactLogix 5370 L3 Controllers and Compact GuardLogix 5370 L3 controllers only Logix Hot Backup - CompactLogix 5370 L3 Controllers only
Integrated motion	EtherNet/IP
Conformal coating	1769-L30ERK, 1769-L30ERMK, 1769-L33ERK, 1769-L33ERMK, 1769-L33ERMSK, 1769-L37ERMK, 1769-L37ERMSK, 1769-L38ERMK, 1769-L38ERMSK

Controller Memory

The Logix CPU executes application code and messages. The backplane CPU transfers I/O memory and other module data on the backplane. This CPU operates independently from the Logix CPU, so it sends and receives I/O information asynchronous to program execution.

TIP CPU usage is based on the number of devices in the I/O tree. About 6% of the CPU is used for every 100 devices in the I/O tree.

1756 ControlLogix 5570 controllers - Memory is separated into isolated sections.

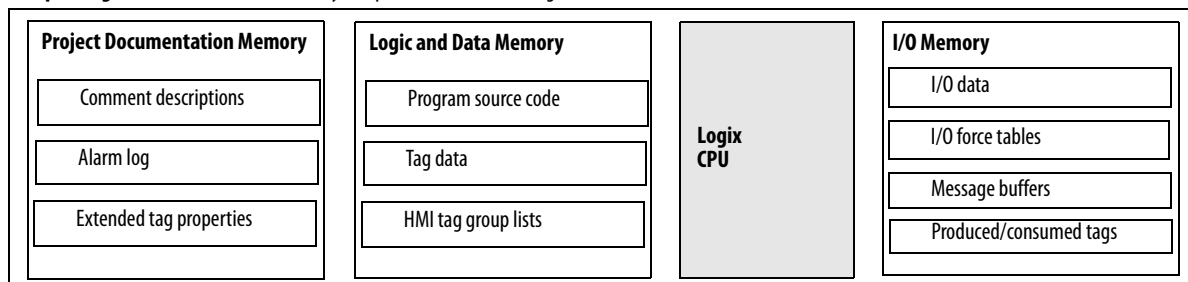


CompactLogix 5370 and Compact GuardLogix 5370 Controllers

The Logix CPU executes application code and messages.

Controller	I/O Task Priority	Communication Task Priority
CompactLogix 5370	6	12

CompactLogix 5370 controllers - Memory is separated into isolated segments.



Controller Connections

The controller uses a connection to establish a communication link between two devices.

IMPORTANT The topics in this section apply only to ControlLogix 5570 and earlier controllers, and CompactLogix 5370 and earlier controllers operation

Connections can be made to the following:

- Controller to local I/O modules or local communication modules
- Controller to remote I/O or remote communication modules
- Controller to remote I/O (rack-optimized) modules
- Produced and consumed tags
- Messages
- Access to programming software
- Linux-based software access for HMI or other software applications

The controllers have different communication limits.

Communication Attribute	1756-L7x ControlLogix	1756-L6x ControlLogix	1769 CompactLogix	CompactLogix 5370	1768 CompactLogix
Connections	500	250	100	256	250
Cached messages	32 for messages and block transfers combined				
Unconnected receive buffers	3				
Unconnected transmit buffers	Default 20 (can be increased to 40)		Default 10 (can be increased to 40)		

The limit of connections can ultimately reside in the communication module you use for the connection. If a message path routes through a communication module, the connection that is related to the message also counts toward the connection limit of that communication module.

Controller	Communication Device	Supported Connections
ControlLogix	1756-CN2R, 1756-CN2RXT	100 CIP™ connections (any combination of scheduled and message connections)
	1756-CN2/B	128 CIP connections
	1756-CNB, 1756 -CNBR	64 CIP connections depending on RPI, recommend that you use only 48 connections (any combination of scheduled and message connections)
	1756-EN2F, 1756-EN2T, 1756-EN2TR, 1756-EN2TP, 1756-EN2TXT, 1756-EN3TR	256 CIP connections 128 TCP/IP connections
	1756-EN4TR	CIP connected messages: • 1000 I/O • 528 ⁽¹⁾ 512 TCP/IP connections
	1756-ENBT 1756-EWEB	128 CIP connections 64 TCP/IP connections
CompactLogix 5370	Built-in Ethernet ports	See the CompactLogix 5370 Controllers User Manual, publication 1769-UM021 , for information on how to count EtherNet/IP nodes on the I/O Configuration section of the programming software.

(1) There are 1000 explicit connections and 528 implicit connections.

Determine Total Connection Requirements

The total connections for a controller include both local and remote connections. Counting local connections is not an issue for CompactLogix controllers. They support the maximum number of modules that are permitted in their systems.

When designing your CompactLogix 5370 controllers, you must consider these resources:

- EtherNet/IP network nodes
- Controller connections

For more information, see the CompactLogix 5370 Controllers User Manual, publication [1769-UM021](#).

The ControlLogix controllers support more communication modules than the other controllers, so you must tally local connections to make sure that you stay within the connection limit.

Use this table to tally **local** connections.

Connection Type	Device Quantity	x	Connections per Module	=	Total Connections
Local I/O module (always a direct connection)		x	1	=	
SERCOS Motion module		x	3	=	
ControlNet communication module		x	0	=	
EtherNet/IP communication module		x	0	=	
DeviceNet communication module		x	2	=	
DH+ /Remote I/O communication module		x	1	=	
DH-485 communication module		x	1	=	
Programming software access to controller		x	1	=	
Total					

IMPORTANT A redundant system uses eight connections in the controller.

The communication modules that you select determine how many remote connections are available. Use this table to tally **remote** connections.

Connection Type	Device Quantity	x	Connections per Module	=	Total Connections
Remote ControlNet communication module Configured as a direct (none) connection Configured as a rack-optimized connection		x	0 or 1	=	
Remote EtherNet/IP communication module Configured as a direct (none) connection Configured as a rack-optimized connection		x	0 or 1	=	
Remote device over a DeviceNet network (accounted for in rack-optimized connection for local DeviceNet module)		x	0	=	
Safety device on a DeviceNet or EtherNet/IP network		x	2	=	
Other remote communication adapter		x	1	=	
Distributed I/O module (individually configured for a direct connection)		x	1	=	
Produced tag and first consumer Each additional consumer		x	2 1	=	
Consumed tag		x	1	=	
Connected message (CIP Data Table read/write and DH+™)		x	1	=	
Block transfer message		x	1	=	
Linux-based software access for HMI or other software applications		x	4	=	
FactoryTalk® Linux software for HMI or other software applications		x	5	=	
Total					

System Overhead Percentage

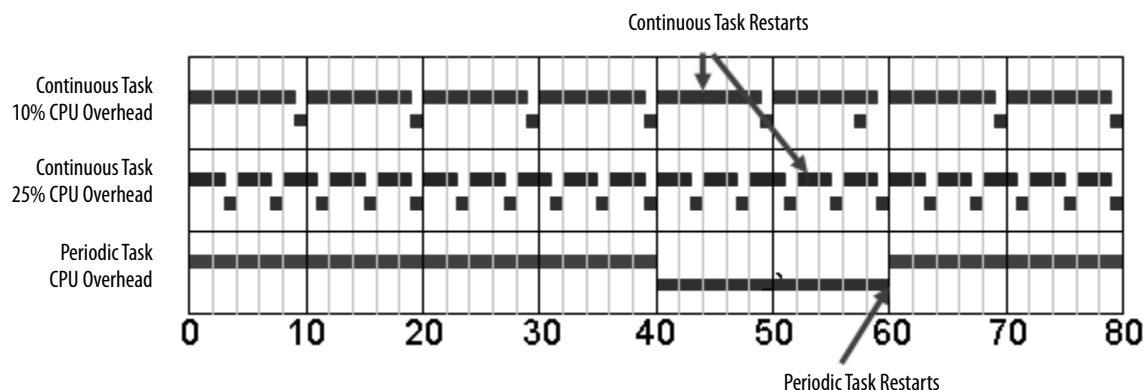
The system overhead timeslice specifies the percentage of continuous task execution time that is devoted to communication and background redundancy functions.

- Message communication is any communication that you do **not** configure through the I/O configuration folder of the project, such as MSG instructions.
- Message communication occurs only when a periodic or event task is not running. If you use multiple tasks, make sure that their scan times and execution intervals leave enough time for message communication.
- System overhead interrupts only the continuous task.
- The controller performs message communication for up to 1 ms at a time and then resumes the continuous task.
- Adjust the update rates of the tasks as needed to get the best trade-off between executing your logic and servicing message communication.

System overhead functions include the following:

- Communicating with HMI devices and programming software
- Sending and responding to messages
- Alarm management processing
- Redundancy qualification

The controller performs system overhead functions for up to 1 ms at a time. If the controller completes the overhead functions in less than 1 ms, it resumes the continuous task. The following chart compares a continuous and periodic task.



Example	Description
Continuous task 10% CPU overhead	In the top example, the system overhead timeslice is set to 10%. Given 40 ms of code to execute, the continuous task completes the execution in 44 ms. During a 60 ms period, the controller is able to spend 5 ms on communication processing.
Continuous task 25% CPU overhead	By increasing the system overhead timeslice to 25%, the controller completes the continuous task scan in 57 ms. The controller spends 15 ms of a 60 ms time span on communication processing.
Periodic task	Placing the same code in a periodic task yields even more time for communication processing. The bottom example assumes that the code is in a 60 ms periodic task. The code executes to completion and then goes dormant until the 60 ms, time-based trigger occurs. While the task is dormant, all CPU bandwidth can focus on communication. Because the code takes only 40 ms to execute, the controller can spend 20 ms on communication processing. Depending on the amount of communication to process during this 20 ms window, it can be delayed as it waits for other modules in the system to process the data that was communicated.

The CPU timeslices between the continuous task and system overhead. Each task switch between user task and system overhead takes additional CPU time to load and restore task information. You can calculate the continuous task interval as:

$$\text{ContinuousTime} = (100 / \text{SystemOverheadTimeSlice\%}) - 1$$

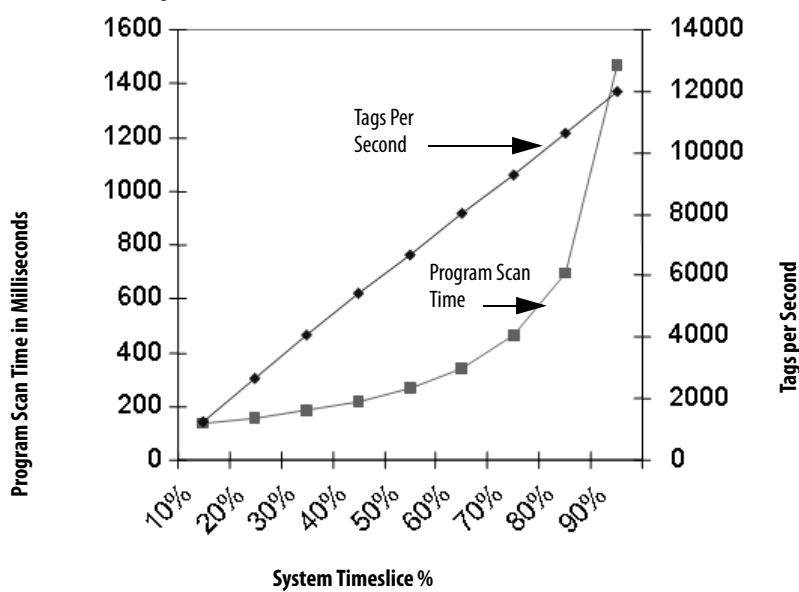
The programming software forces at least 1 ms of execution time for the continuous task, regardless of the system overhead timeslice. This more efficiently uses system resources because letting shorter execution times of the continuous task exist means switching tasks more frequently.

System Overhead Timeslice %	Communication Execution (ms)	Continuous Task Execution (ms)
10	1	9
20	1	4
33	1	2
50	1	1
66	2	1
80	4	1
90	9	1

Manage the System Overhead Timeslice Percentage

As the system overhead timeslice percentage increases, time that is allocated to executing the continuous task decreases. If there is no communication for the controller to manage, the controller uses the communication time to execute the continuous task.

IMPORTANT System Overhead Time Slice does not apply to ControlLogix 5580 or CompactLogix 5380 controllers.

Consideration	Description																														
Continuous task always has at least 1 ms execution time	The programming software forces the continuous task to have at least 1 ms of execution time, regardless of the setting for the system overhead timeslice. This results in more efficient controller use because excessive swapping between tasks uses valuable CPU resources.																														
Impact on communication and scan time	<p>Increasing the system overhead timeslice percentage decreases execution time for the continuous task while it increases communication performance.</p> <p>Increasing the system overhead timeslice percentage also increases the amount of time it takes to execute a continuous task - increasing overall scan time.</p>  <table><caption>Approximate data points from the graph</caption><thead><tr><th>System Timeslice %</th><th>Program Scan Time (ms)</th><th>Tags Per Second</th></tr></thead><tbody><tr><td>10%</td><td>150</td><td>1000</td></tr><tr><td>20%</td><td>180</td><td>2000</td></tr><tr><td>30%</td><td>220</td><td>3000</td></tr><tr><td>40%</td><td>280</td><td>4000</td></tr><tr><td>50%</td><td>350</td><td>5000</td></tr><tr><td>60%</td><td>450</td><td>6000</td></tr><tr><td>70%</td><td>550</td><td>7000</td></tr><tr><td>80%</td><td>700</td><td>8000</td></tr><tr><td>90%</td><td>1450</td><td>13000</td></tr></tbody></table>	System Timeslice %	Program Scan Time (ms)	Tags Per Second	10%	150	1000	20%	180	2000	30%	220	3000	40%	280	4000	50%	350	5000	60%	450	6000	70%	550	7000	80%	700	8000	90%	1450	13000
System Timeslice %	Program Scan Time (ms)	Tags Per Second																													
10%	150	1000																													
20%	180	2000																													
30%	220	3000																													
40%	280	4000																													
50%	350	5000																													
60%	450	6000																													
70%	550	7000																													
80%	700	8000																													
90%	1450	13000																													
Unused portion of system overhead timeslice	<p>You can configure any unused portion of the system overhead timeslice to:</p> <ul style="list-style-type: none">Run the continuous task, which results in faster execution of application code and increases the variability of the program scan.Process communication, which results in more predictable and deterministic scan time for the continuous task. (This is for development and testing of an application to simulate communication.)																														
System overhead	<p>System overhead is the time that the controller spends on message communication and background tasks.</p> <ul style="list-style-type: none">Message communication is any communication that you do not configure through the I/O configuration folder of the project, such as MSG instructions.Message communication occurs only when a periodic or event task is not running. If you use multiple tasks, make sure that their scan times and execution intervals leave enough time for message communication.System overhead interrupts only the continuous task.The system overhead timeslice specifies the percentage of time (excluding the time for periodic or event tasks) that the controller devotes to message communication.System overhead timeslice does not apply to ControlLogix 5580 and CompactLogix 5380 controllers.The controller performs message communication for up to 1 ms at a time and then resumes the continuous task.Adjust the update rates of the tasks as needed to get the best trade-off between executing your logic and servicing message communication.																														

Individual applications can differ, but the overall impact on communication and scan time remains the same. The data is based on a ControlLogix5555 controller running a continuous task with 5000 tags (no arrays or user-defined structures).

I/O Processing

The 5370 controllers use a dedicated periodic task to process I/O data. This I/O task:

- Operates at priority 6.
- Higher-priority tasks take precedence over the I/O task and can affect processing.
- Executes at the fastest RPI you have scheduled for the system.
- Executes for as long as it takes to scan the configured I/O modules.

Data Types

The controllers support IEC 61131-3 atomic data types. The controllers also support compound data types, such as arrays, predefined structures (such as counters and timers), and user-defined structures.

The Logix CPU reads and manipulates 32-bit data values. The minimum memory allocation for data in a tag is 4 bytes. When you create a standalone tag that stores data that is less than 4 bytes, the controller allocates 4 bytes, but the data only fills the part that it needs.

For more information See [Data Structures on page 75](#).

Data Type	Bits							
	64...32	31	16	15	8	7	1	0
BOOL	Not allocated	Allocated but not used						0 or 1
SINT	Not allocated	Allocated but not used					-128...+127	
INT	Not allocated	Allocated but not used			-32,768...32,767			
DINT	Not allocated	-2,147,483,648...2,147,483,647						
REAL	Not allocated	-3.40282347E ³⁸ ...-1.17549435E ⁻³⁸ (negative values) 0 1.17549435E ⁻³⁸ ...3.40282347E ³⁸ (positive values)						
LINT	Valid Date/Time range is from 1/1/1970 12:00:00 AM coordinated universal time (UTC) to 1/1/3000 12:00:00 AM UTC							

Programming Techniques

Programming Technique	Consideration
Subroutines	For Studio 5000 Logix Designer® Version 28 and later on 5570 and 5370 controllers: <ul style="list-style-type: none"> JSR calls are limited to 40 input parameters and 40 output parameters. There is no limit on nesting JSR instructions. However, it is possible that too many nesting levels can cause the controller to run out of memory and fault.
Add-On Instructions	For 5570 controllers or earlier, and 5370 controllers or earlier, there is no limit on nesting Add-On Instructions. However, it is possible that too many nesting levels can cause the controller to run out of memory and fault.

For more information See [Modular Programming Techniques on page 45](#).

Produced and Consumed Data

The controller supports:

- Total number of produced tags ≤ 127
- Maximum number of multicast produce tags out of the CompactLogix Ethernet port ≤ 32
- Maximum number of consumed tags ≤ 250 (or controller maximum)

For more information See [Produced and Consumed Data on page 71](#)

Messages

The controller supports:

- As many outgoing, unconnected buffers as fit in controller memory.

Each buffer uses approximately 1.2 KB of I/O memory.

You can use a CIP Generic message instruction to increase the number of unconnected buffers. See the Logix 5000™ Controllers Messages Programming Manual, publication [1756-PM012](#).

- Three incoming unconnected buffers
- 32 cached buffers, as of firmware revision 12 and later.

Logic Execution

The controller operating system is a pre-emptive multitasking system that is IEC 61131-3 compliant.

Tasks to configure controller execution	A task provides scheduling and priority information for a set of one or more programs. You can configure tasks as either continuous, periodic, or event.
Programs to group data and logic	A task contains programs, each with its own routines and program-scoped tags. Once a task is triggered (activated), the programs that are assigned to the task execute in the order in which they are listed in the Controller Organizer. Programs are useful for projects that multiple programmers develop. During development, the code in one program that uses program-scoped tags can be duplicated in a second program to minimize the possibility of tag names colliding. Tasks can contain programs and equipment phases.
Routines to encapsulate executable code that is written in one programming language	Routines contain the executable code. Each program has a main routine that is the first routine to execute within a program. Use logic, such as the Jump to Subroutine (JSR) instruction, to call other routines. You can also specify an optional program fault routine.

Decide When to Use Tasks, Programs, and Routines

Use these considerations to determine when to use a task, program, or routine.

Comparison	Task	Program and Equipment Phase	Routine
Quantity available	Varies by controller (4, 6, 8, or 32)	32 program and equipment phases (combined) per task 100 for ControlLogix® controllers with V23 and earlier 1000 programs/task for ControlLogix controllers with V24 and later	Unlimited number of routines per program
Function	Determines how and when code is executed	Organizes groups of routines that share a common data area and function.	Contains executable code (relay ladder, function block diagram, sequential function chart, or structured text)
Use	<ul style="list-style-type: none"> Most code is expected to reside in a continuous task Use a periodic task for slower processes or when time-based operation is critical Use an event task for operations that require synchronization to a specific event 	<ul style="list-style-type: none"> Put major equipment pieces or plant cells into isolated programs Use programs to isolate different programmers or create reusable code Configurable execution order within a task Isolate individual batch phases or discrete machine operations 	<ul style="list-style-type: none"> Isolate machine or cell functions in a routine Use the appropriate language for the process Modularize code into subroutines that can be called multiple times
Considerations	<ul style="list-style-type: none"> A high number of tasks can be difficult to debug Can disable output processing on some tasks to improve performance Tasks can be inhibited to help prevent execution Do not configure multiple tasks at the same priority 	<ul style="list-style-type: none"> Data spanning multiple programs must be controller-scoped or a program parameter. Listed in the Controller Organizer in the order of execution 	<ul style="list-style-type: none"> Subroutines with multiple calls can be difficult to debug Data can be referenced from program-scoped and controller-scoped areas Calling many routines impacts scan time Listed in the Controller Organizer as Main, Fault, and then alphabetically

Specify Task Priorities

Each task in the controller has a priority level. A higher priority task (such as 1) interrupts any lower priority task (such as 15). The continuous task has the lowest priority; periodic or event tasks always interrupt continuous tasks.

The controller has these types of tasks.

Priority	User Task	Description
Highest	—	CPU overhead - general CPU operations
	—	Motion planner - executed at coarse update rate
	—	Safety task - safety logic
	—	Redundancy task - communication in redundant systems
	—	Trend data collection - high-speed collection of trend data values
	Priority 1 Event/Periodic	User defined
	Priority 2 Event/Periodic	User defined
	Priority 3 Event/Periodic	User defined
	Priority 4 Event/Periodic	User defined
	Priority 5 Event/Periodic	User defined
	Priority 6 Event/Periodic	User defined 1769 CompactLogix™ controllers process I/O as a periodic task based on the chassis RPI setting
	Priority 7 Event/Periodic	User defined
	Priority 8 Event/Periodic	User defined
	Priority 9 Event/Periodic	User defined
	Priority 10 Event/Periodic	User defined
Lowest	Priority 11 Event/Periodic	User defined
	Priority 12 Event/Periodic	User defined CompactLogix communication and scheduled connection maintenance
	Priority 13 Event/Periodic	User defined
	Priority 14 Event/Periodic	User defined
	Priority 15 Event/Periodic	User defined
	Continuous	Lowest priority task. Interrupted by all other tasks including system overhead timeslice (if applicable.)

If a periodic or event task is executing when another is triggered, and both tasks are at the same priority level, the tasks execute in 1 ms increments until one of the tasks completes execution.

Manage User Tasks

You can configure these user tasks.

If you want logic to execute	Use this task	Description
All of the time	Continuous task	The continuous task runs in the background. Any CPU time that is not allocated to other operations or tasks is used to execute the continuous task. <ul style="list-style-type: none"> The continuous task runs all of the time. When the continuous task completes a full scan, it restarts immediately. A project does not require a continuous task. If used, there can be only one continuous task.
<ul style="list-style-type: none"> At a constant period (such as every 100 ms) Multiple times within the scan of your other logic 	Periodic task	A periodic task performs a function at a specific time interval. Whenever the time for the periodic task expires, the periodic task: <ul style="list-style-type: none"> Interrupts any lower priority tasks. Executes one time. Returns control to where the previous task left off.
Immediately when an event occurs	Event task	An event task performs a function only when a specific event (trigger) occurs. Whenever the trigger for the event task occurs, the event task: <ul style="list-style-type: none"> Interrupts any lower priority tasks. Executes one time. Returns control to where the previous task left off.

The user tasks that you create appear in the Tasks folder of the controller. The predefined system tasks do not appear in the Tasks folder and they do not count toward the task limit of the controller.

Pre-defined Tasks in ControlLogix and CompactLogix Process Controllers

PlantPAx system release 5.0 adds process controllers to the Logix 5000 family of controllers. The process controllers offer additional capabilities that are targeted for DCS applications.

The Task folder contains a project structure that consists of four pre-defined periodic tasks:

- Fast (100 ms) – For control fast loops, such as liquid flow or pressure with related transmitters and pump drives
- Normal (250 ms) – For discrete control, such as motors, pumps, and valves
- Slow (500 ms) – For level, temperature, analysis loops, phases, and batch sequencing
- System (1000 ms) – For slow change temperature control and general controller operations, such as messaging or status

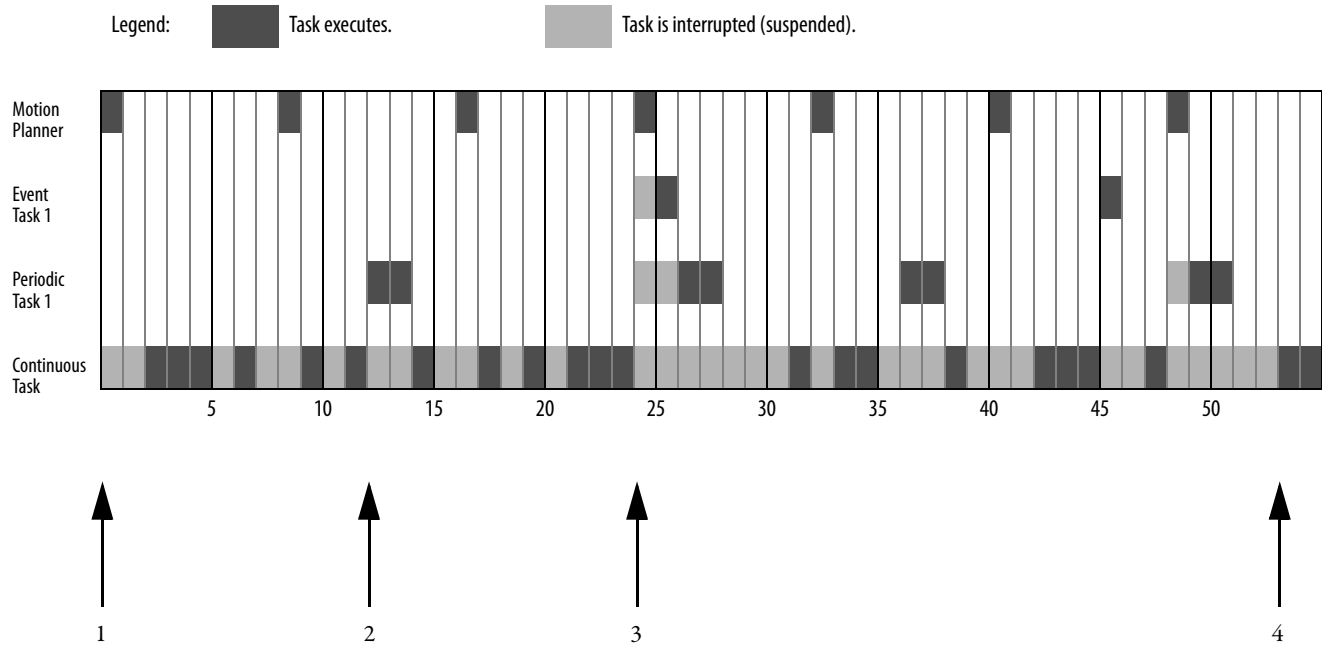
Considerations that Affect Task Execution

Consideration	Description
Motion planner	The motion planner interrupts all other tasks, regardless of their priority. <ul style="list-style-type: none"> The number of axes and coarse update period for the motion group affect how long and how often the motion planner executes. If the motion planner is executing when a task is triggered, the task waits until the motion planner is done. If the coarse update period occurs while a task is executing, the task pauses to let the motion planner execute.
Output processing	At the end of a task, the controller performs output processing for the output modules in your system. This processing depends on the number of output connections that are configured in the I/O tree.
Too many tasks	If you have too many tasks, then the following can occur: <ul style="list-style-type: none"> Continuous task can take too long to complete. Other tasks can experience overlaps. If a task is interrupted too frequently or too long, it must be triggered again to complete its execution. Controller communication can be slower. If your application is designed for data collection, try to avoid multiple tasks.

This example depicts the execution of a project with these tasks.

Table 1 - Example Task Execution

Task	Priority	Period	Execution Time	Duration
Motion planner	—	8 ms (course update rate)	1 ms	1 ms
Event task 1	1	—	1 ms	1 . . 2 ms
Periodic task 1	2	12 ms	2 ms	2 . . 4 ms
Continuous task	—	—	20 ms	48 ms



Description	
1	Initially, the controller executes the motion planner and the I/O task (if one exists).
2	The period for periodic task 1 expires (12 ms), so the task interrupts the continuous task.
3	The triggers occur for event task 1. Event task 1 waits until the motion planner is done. Lower priority tasks experience longer delays.
4	The continuous task automatically restarts.

Configure a Continuous Task

When you create a project in the programming software, the Main Task is configured as a continuous task.

- A controller supports one continuous task, but a continuous task is not required.
- You can configure the task to update output modules at the end of the continuous task.
- You can change the continuous task to either a periodic or event task.

Configure a Periodic Task

A periodic task executes automatically based on a preconfigured interval. You can configure whether the task updates output modules at the end of the periodic task. After the task executes, it does not execute again until the configured time interval has elapsed.

Configure an Event Task

An event task executes automatically based on a trigger event occurring, or if a trigger event does not occur in a specific time interval. You configure whether the task updates output modules at the end of the task. After the task executes, it does not execute again until the event occurs again. Each event task requires a specific trigger.

Trigger	Description
Module Input Data State Change	A remote input module (digital or analog) triggers an event task that is based on the change of state (COS) configuration for the module. Enable COS for only one point on the module. If you enable COS for multiple points, a task overlap of the event task can occur. <ul style="list-style-type: none"> • The ControlLogix sequence of events modules (1756-IB16ISOE, 1756-IH16ISOE) use the Enable Coordinated System Time (CST) Capture feature instead of COS. • The embedded input points on the 1769-L16ER-BB1B, 1769-L18ER-BB1B, 1769-L18ERM-BB1B, 1769-L18ERM-BB1B, and 1769-L19ER-BB1BK modules can be configured to trigger an event task when a COS occurs.
Consumed Tag	Only one consumed tag can trigger a specific event task. Use an Immediate Output (IOT) instruction in the producing controller to signal the production of new data.
Axis Registration 1 or 2	A registration input triggers the event task.
Axis Watch	A watch position triggers the event task.
Motion Group Execution	The coarse update period for the motion group triggers the execution of both the motion planner and the event task. Because the motion planner interrupts all other tasks, it executes first.
EVENT Instruction	Multiple EVENT instructions can trigger the same task.

For more information on event tasks, see:

- Logix 5000™ Controllers Common Procedures Programming Manual, publication [1756-PM001](#)
- Using Event Tasks with Logix 5000 Controllers, publication [LOGIX-WP003](#)

Guidelines to Configure an Event Task

Guideline	Description
Place the I/O module being used to trigger an event in the same chassis as the controller.	Placing the I/O module in a remote chassis adds more network communication and processing to the response time.
Limit events on digital inputs to one input bit on a module.	All inputs on a module trigger one event, so if you use multiple bits you increase the chance of a task overlap. Configure the module to detect change of state on the trigger input and turn off the other bits.
Set the priority of the event task as the highest priority on the controller.	If the priority of the event task is lower than a periodic task, the event task has to wait for the periodic task to complete execution.
Limit the number of event tasks.	Increasing the number of event tasks reduces the available CPU bandwidth and increases the chances of task overlap.

Additional Considerations for Periodic and Event Tasks

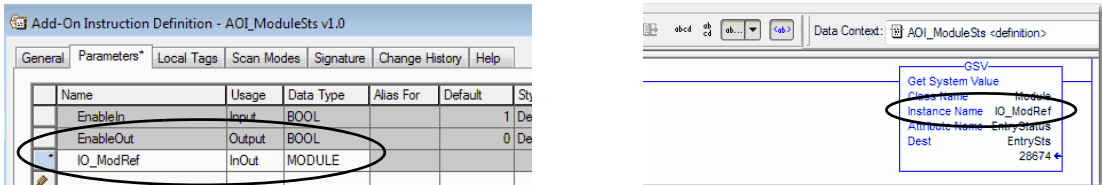
Consideration	Description
Amount of code in the event task	Each logic element (for example, rung, instruction, or structured text construct) adds to scan time.
Task priority	If the event task is not the highest priority task, a higher priority task can delay or interrupt the execution of the event task.
CPS and UID instructions	If one of these instructions is active, the event task cannot interrupt the currently executing task. (The task with the CPS or UID.)
Motion planner	The motion planner takes precedence over event or periodic tasks
Trends	Trend data collection takes precedence over event or periodic tasks.
Output processing	You can disable output processing at the end of a task to reduce the amount of task processing time. The Controller Organizer displays whether outputs processing is disabled.

Access the Module Object

The MODULE object provides status information about a module. To select a particular module object, set the Object Name operand of the GSV/SSV instruction to the module name. The specified module must be present in the I/O Configuration section of the controller organizer and must have a device name.

With the Studio 5000 Logix Designer® application, version 24.00.00 and later, you can access the MODULE object directly from an Add-On Instruction. Previously, you could access the MODULE object data, but not from within an Add-On Instruction.

You must create a **Module Reference** parameter when you define the Add-On Instruction to access the MODULE object data. A Module Reference parameter is an InOut parameter of the MODULE data type that points to the MODULE Object of a hardware module. You can use module reference parameters in both Add-On Instruction logic and program logic.



For more information on the Module Reference parameter, see the Logix 5000 Controllers Add-On Instructions Programming Manual, publication [1756-PM010](#), and the Logix Designer application online help.

The **Path** attribute is available with Logix Designer application, version 24.00.00 and later. This attribute provides a communication path to the module.

For more information on the attributes available in the MODULE object, see the Logix 5000 Controllers General Instructions Reference Manual, publication [1756-RM003](#).

Develop Application Code in Routines

Each routine contains logic in one programming language. Choose a programming language that is based on the application.

Section of Code Represents	Language to Use
Continuous or parallel execution of multiple operations (not sequenced)	Ladder Diagram(LD)
Boolean or bit-based operations	
Complex logical operations	
Message and communication processing	
Machine interlocking	
Operations that service or maintenance personnel can interpret to troubleshoot the machine or process.	
Servo motion control	
Continuous process and drive control	Function block diagram (FBD)
Loop control	
Calculations in circuit flow	
High-level management of multiple operations	Sequential function chart (SFC)
Repetitive sequences of operations	
Batch process	
Motion control sequencing (via sequential function chart with embedded structure text)	
State machine operations	
Complex mathematical operations	Structured text (ST)
Specialized array or table loop processing	
ASCII string handling or protocol processing	

Comparison of Programming Languages

Comparison	Relay Ladder Logic	Function Block Diagram	Sequential Function Chart	Structured Text
Instruction categories	<ul style="list-style-type: none"> Boolean General and trig math Timers and counters Array management Diagnostic Serial port and messaging ASCII manipulation Motion control 	<ul style="list-style-type: none"> General and trig math Timers and counters Bitwise logical Advanced process Advanced drive 	<ul style="list-style-type: none"> Step/action with embedded structured text Transition with structure text comparisons Simultaneous and selection branches Stop element 	<ul style="list-style-type: none"> General and trig math Timers and counters Bitwise logical Array management Diagnostic ASCII manipulation Specialty CPU control Motion control Advanced process Advanced drive
Editor style	<ul style="list-style-type: none"> Graphical rungs Unlimited rungs 	<ul style="list-style-type: none"> Graphical, free-form drawing Unlimited sheets 	<ul style="list-style-type: none"> Graphical, free-form drawing Unlimited grid space 	<ul style="list-style-type: none"> Textual Unlimited lines
Monitoring	<ul style="list-style-type: none"> Rung animation Data value animation Force status 	<ul style="list-style-type: none"> Output and input pin data value animation 	<ul style="list-style-type: none"> Active steps animation Auto display scroll Branch/transition force status 	<ul style="list-style-type: none"> Tag watch pane Context coloring
Comments	<ul style="list-style-type: none"> Tag Rung 	<ul style="list-style-type: none"> Tag Text box 	<ul style="list-style-type: none"> Tag Text box Embedded structured text comments that are stored in CPU 	<ul style="list-style-type: none"> Multi-line End if line Comments that are stored in CPU

Programming Methods

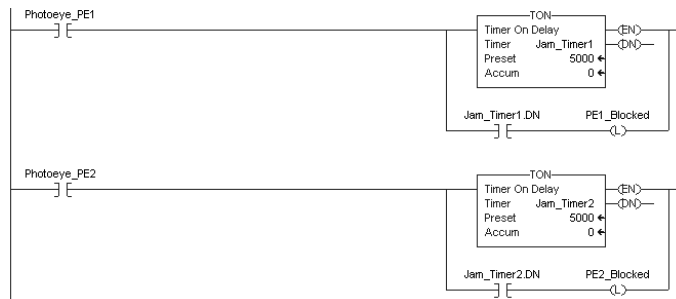
The capabilities of the controllers make different programming methods possible. There are trade-offs to consider when selecting a programming method.

Inline Duplication

Write multiple copies of the code with different tag references.

Benefits

- Uses more memory
- Fastest execution time because all tag references are defined before runtime
- Easiest to maintain because rung animation matches tag values
- Requires more time to create and modify

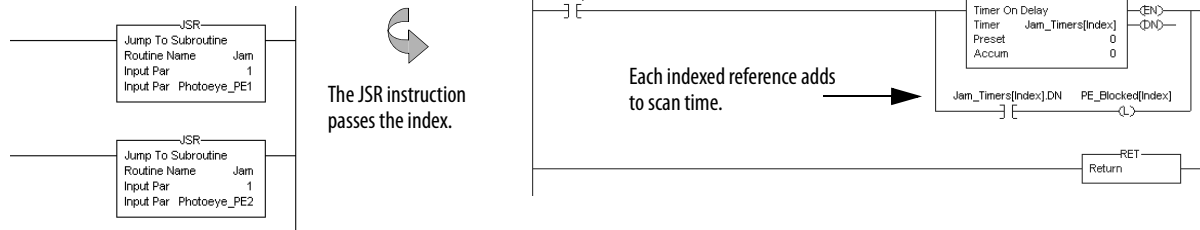


Indexed Routine

Write one copy of code and use indexed references to data stored in arrays.

Benefits

- One copy of code is faster to develop
- Slowest execution time because all tag references are calculated at runtime
- Can be difficult to maintain because the data monitor is not synchronized to execution

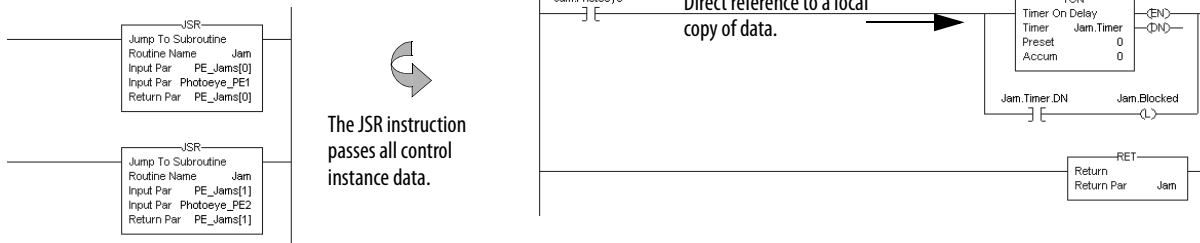


Buffered Routine

Copy the values of an array into tags to directly reference these buffer tags.

Benefits

- One copy operation can occur faster than multiple index offsets
- Minimizes the need to calculate array offsets at runtime
- The amount of code increases, but so do the benefits
- Can be difficult to maintain because the data monitor is not synchronized to execution



Controller Prescan of Logic

On transition to Run mode, the controller prescans logic to initialize instructions. The controller resets all state-based instructions, such as outputs (OTE) and timers (TON). Some instructions also perform operations during prescan. For example, the ONSR instruction turns off the storage bit. For information on prescan, see the following resources:

- Logix 5000 Controllers General Instructions Reference Manual, publication [1756-RM003](#).
- Logix 5000 Controllers Process Control and Drives Instructions Reference Manual, publication [1756-RM006](#).

During prescan, input values are not current and outputs are not written.

Prescan Affects	Description
Relay ladder logic	The controller resets non-retentive I/O and internal values.
Function block diagram logic	Along with resetting non-retentive I/O and internal values, the controller also clears the EnableIn parameter for every function block diagram.
Structured text logic	The controller resets bit tags and forces numeric tags to zero . Use the bracketed assignment operator ([:=]) to force a value to be reset during prescan. If you want a tag that is left in its last state, use the non-bracketed assignment operator (:=).
Sequential function chart logic	Embedded structured text follows the same rules as listed previously.

Prescan differs from first scan in that the controller does not execute logic during prescan. The controller executes logic during first scan. The controller sets S:FS for one scan:

- During the first scan that follows prescan.
- During the first scan of a program when it has been uninhibited.
- Each time a step is first scanned (when step.FS is set). You can view the S:FS bit being set only from the logic that is contained in actions that execute during the first scan of their parent step (N, L, P, and P1).

Add-On Instruction Prescan Logic

An Add-On Instruction prescan logic executes after the main logic executes in Prescan mode. Use the prescan logic to initialize tag values before execution. For example, set a PID instruction to Manual mode with a 0% output before its first execution.

When an Add-On Instruction executes in Prescan mode, any required parameters have their data passed.

- Values are passed to Input parameters from their arguments in the instruction call.
- Values are passed out of Output parameters to their arguments defined in the instruction call.

Controller Postscan of SFC Logic

SFCs support an automatic reset option that performs a postscan of the actions that are associated with a step once a transition indicates that the step is completed. Also, every Jump to Subroutine (JSR) instruction causes the controller to postscan the called routine. During this postscan:

- Output energize (OTE) instructions are turned off and non-retentive timers are reset.
- In structured text code, use the bracketed assignment operator (`[:=]`) to have tags reset.
- In structured text code, use the non-bracketed assignment operator (`:=`) to have tags that are left in their last state.
- Selected array faults, that is, 4/20 and 4/83, can be suppressed. When the fault is suppressed, the controller uses an internal fault handler to clear it. Clearing the fault causes the postscan process to skip the instruction containing the fault and continue with the next instruction. This occurs only when SFC instructions are configured for automatic reset.

Add-On Instruction Postscan Logic

When an Add-On Instruction is called by logic in an SFC Action and the Automatic Reset option is set, the Add-On Instruction executes in Postscan mode. An Add-On Instruction postscan routine executes after the main logic executes in Postscan mode. Use the postscan logic to reset internal states and status values or to disable instruction outputs when the SFC action completes.

Timer Execution

Timers in the controllers store off a portion of the real-time clock each time they are scanned. The next time through, they compare this stored value against the current clock and then adjust the ACC value by the difference.

The controller uses native 32-bit data, so there is more space to store the time. The timer stores 22 bits at 1 ms/bit, which equates to 69.905 minutes ($2^{22} / 1000 \text{ ms per second} / 60 \text{ seconds per minute}$) of padding before a timer overlaps.

If program execution skips timers, it appears as if the timers pause. Actually, the timers are overrunning themselves. Depending on when the timer logic next executes, the lost time varies ranges from 0...69.905 minutes.

Program execution can skip executing timers due to the following:

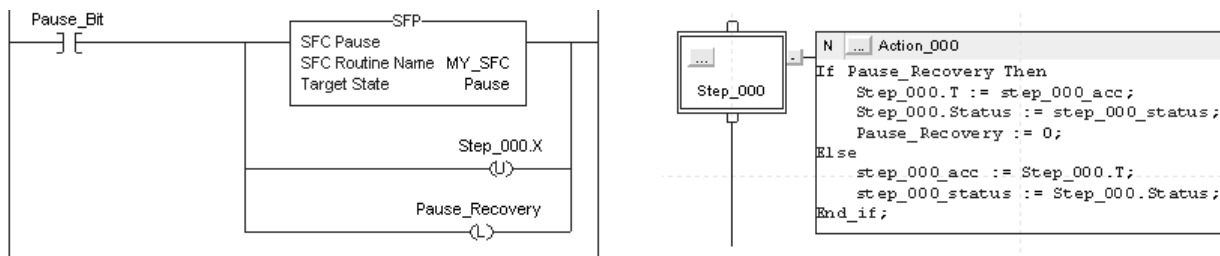
- Subroutine not being called
- Jumping over code
- SFC action
- Inactive SFC step
- Event or periodic task not executing
- Equipment phase state routines

SFC Step Timer Execution

An SFC step timer stores the clock time each time the step executes. On subsequent scans of the step, the controller compares the current clock time with the last scan and updates the step timer's ACC by the difference.

When you pause an SFC and then release the SFC, the step timer jumps forward by the duration of the pause. If you want a step timer to remain at its position during a pause:

- Latch a recovery bit when the chart pause is released.
- Add an action to the step to store the step timer's ACC value and restore that value when the pause recovery bit is set.



Edit an SFC Online

When you edit an SFC online, the software initially makes the changes in the offline project. When you accept the changes, they are downloaded to the controller. If you transition the controller to test or untest edits, the controller resets the SFC and starts execution at the initial step. If you edit an SFC online, do the following:

- Plan when you test or untest edits to coincide with the SFC executing the initial step.
- Place structured text logic in subroutines to minimize the impact of online edits.
- Use an SFR instruction to shift SFC execution to the desired step programmatically.

In some cases, this can result in the SFC being out of sync with the equipment. Program logic in the initial step to check the last state and use an SFR instruction to change to the appropriate step, if needed. One method is to set an index number in an action of each step. Then when the restart occurs, use the SFR instruction to jump to appropriate step based on the index value.

As of firmware revision 18, the following online edits to an SFC no longer reset the SFC to the initial step:

- Modified structured text in actions and transitions
- Physically moved steps, actions, and transitions on SFC sheets without changing the wiring
- Added, deleted, or modified text and description boxes
- Modified indicator tags

Notes:

Modular Programming Techniques

Modular programming guidelines support the delivery of standardized programming structures, conventions, configurations, and strategies. The goal of modular programming is to provide consistency.

- Faster and easier development of application software
- Faster and easier testing of application software
- More reliable application software
- Improved maintenance and operation of application software
- Improved interoperability with other equipment and systems

This chapter applies to these controllers, and where applicable, the controllers are known as:

Controller Family	Includes these controllers
5580 controllers	ControlLogix® 5580 and GuardLogix® 5580 controllers
5380 controllers	CompactLogix™ 5380 and Compact GuardLogix 5380 controllers
5570 controllers	ControlLogix 5570 and GuardLogix 5570 controllers
5370 controllers	CompactLogix 5370 and Compact GuardLogix 5370 controllers
5480 Controllers	CompactLogix 5480 Controllers

Guidelines for Code Reuse

Guideline	Description
Use user-defined data types (UDTs) to group data.	<p>Within a UDT:</p> <ul style="list-style-type: none"> You can mix data types. The tag names that you assign self-document the structure.
Use Add-On Instructions to create standardized modules of code for reuse across a project.	<p>Use an Add-On Instruction to:</p> <ul style="list-style-type: none"> Encapsulate specific or focused operations, such as a Motor or Valve action. A Conveyor or Tank action is better managed as a routine. Create extensions to the base controller instructions. For example, create an Add-On Instruction to execute an SLC™ 500 or PLC controller instruction not available in the Logix 5000™ controllers. Encapsulate an instruction from one language for use in another language. For example, create a function block PIDE instruction for use in relay ladder.
Use program parameters to share data between programs.	<p>Program parameters:</p> <ul style="list-style-type: none"> Are publicly accessible outside of the program. Support external HMI external access on an individual basis for each parameter. <p>Direct access lets the user reference program parameters in logic without configuring parameters in the local program. For example, if Program A has an output parameter that is called Tank_Level, Program B can reference the Tank_Level parameter in logic without creating a corresponding parameter to connect to Program A.</p>
Use partial import/export programs, routines, Add-On Instructions, and code segments to create libraries of reusable code.	<p>Partial import and export of routines and programs:</p> <ul style="list-style-type: none"> Provides more control over the scope of what is extracted from the project. Provides reusable code for larger machine, cell, or unit control. Promotes collaboration between multiple engineers, code standardization, and reuse. <p>The export .LSX file includes all pertinent information, including program configuration, code, user-defined data types, tags, and descriptions, in an XML-formatted, ASCII text file. Use partial import/export to:</p> <ul style="list-style-type: none"> Distribute code separately from the project .ACD file. Edit and create programs and routines by using other editing tools.
Use subroutines to reuse code within a program.	<p>Subroutines:</p> <ul style="list-style-type: none"> Can be created and used in standard and safety applications. Pass User-Defined Structures (UDT). Pass all input and output Parameters by value. Subroutines require the most overhead to pass parameters when called. Can only be called from within the program they reside.

Naming Conventions

The following conventions are guidelines to help make an engineering library more reusable by other developers. These guidelines also help the resulting applications have a more consistent look and feel.

- Names that are meaningful (and readable) to people who use the application as a later date are most effective.
- Names use controller memory and have limited length, so keep them short by using abbreviations and acronyms. Use mixed case rather than underscore characters to indicate words.
- When you use acronyms, use those that are common or provided by industry standards.

Names for controller logic components must follow these guidelines.

- The name must start with a letter, either upper or lower case
- The name can contain as many as 40 characters; any mix of upper case letter, lower case letters, numbers, and underscore characters
- Case is not significant. The controller interprets Mix_Tank the same and mix_tank. However, the software displays the case as entered
- Underscores are significant. The controller interprets AB_CD as unique from A_BCD
- You cannot have two or more underscore characters in a row
- The name cannot end with an underscore.

Component Name	Recommendations														
Controller	<p>Area, unit, or units the controller controls, underscore, type of controller</p> <p>Example:</p> <table> <tr> <td>Area/Unit + Type</td><td>Controller Name: Mixing:ControlLogix</td></tr> </table>	Area/Unit + Type	Controller Name: Mixing:ControlLogix												
Area/Unit + Type	Controller Name: Mixing:ControlLogix														
Controller project	<p>Controller name, the letter C, 1-digit major revision number, underscore, 2-digit minor revision number</p> <p>Example:</p> <table> <tr> <td>Project in controller Mixing_CLX, Major Revision 1, Minor Revision 02</td><td>Application Name: Mixing_CLX_C2_092.ACD</td></tr> </table> <p>Increment the minor revision number for any documented engineering change according to the code in the controller (for example, the code for minor process or equipment changes).</p> <p>Increment the major revision number for any documented engineering change according to the code in the controller that implements a design change (for example, code that enhances or reduces controller functionality).</p>	Project in controller Mixing_CLX, Major Revision 1, Minor Revision 02	Application Name: Mixing_CLX_C2_092.ACD												
Project in controller Mixing_CLX, Major Revision 1, Minor Revision 02	Application Name: Mixing_CLX_C2_092.ACD														
Tag	<p>Prefix with the abbreviation of the type of tag</p> <p>Examples:</p> <table> <tr> <td>Interprocessor communication tag</td><td>IPC_</td></tr> <tr> <td>Input tag</td><td>I_</td></tr> <tr> <td>Output tag</td><td>O_</td></tr> <tr> <td>Remote I/O tag</td><td>RIO_</td></tr> <tr> <td>Control module class tag</td><td>Device ID_</td></tr> <tr> <td>Equipment module class tag</td><td>EM_</td></tr> <tr> <td>Equipment phase class tag</td><td>EP_</td></tr> </table>	Interprocessor communication tag	IPC_	Input tag	I_	Output tag	O_	Remote I/O tag	RIO_	Control module class tag	Device ID_	Equipment module class tag	EM_	Equipment phase class tag	EP_
Interprocessor communication tag	IPC_														
Input tag	I_														
Output tag	O_														
Remote I/O tag	RIO_														
Control module class tag	Device ID_														
Equipment module class tag	EM_														
Equipment phase class tag	EP_														

Component Name	Recommendations																																						
I/O or communication module	<p>Controller name, underscore, abbreviation of rack location (L=local, R=remote), underscore, the letter S, 2-digit slot number, underscore, abbreviation of function</p> <p>Example Functions:</p> <table> <tr><td>Analog input</td><td>AI</td></tr> <tr><td>Analog output</td><td>AO</td></tr> <tr><td>Discrete input</td><td>DI</td></tr> <tr><td>Discrete output</td><td>DO</td></tr> <tr><td>Analog input/output combination</td><td>AIO</td></tr> <tr><td>Discrete input/output combination</td><td>DIO</td></tr> <tr><td>Analog/discrete input/output combination</td><td>ADIO</td></tr> <tr><td>Serial data</td><td>SIO</td></tr> <tr><td>Motion data</td><td>MIO</td></tr> <tr><td>DeviceNet® data</td><td>DNET</td></tr> <tr><td>EtherNet/IP™ data</td><td>ENET</td></tr> <tr><td>ControlNet®</td><td>CNET</td></tr> </table> <p>Examples:</p> <table> <tr> <td>Mixer123 Controller, Local chassis, Slot 4, Analog Output</td><td>Module Name: M123_CLX_L00_S04_AO</td></tr> <tr> <td>Mixer123 Controller, Local chassis, Slot 12, Discrete Output</td><td>Module Name: M123_CLX_L00_S12_DO</td></tr> <tr> <td>Mixer123 Controller, Remote chassis #1, Slot 1, Analog Input</td><td>Module Name: M123_CLX_R01_S01_AI</td></tr> <tr> <td>Mixer123 Controller, Remote chassis #1, Slot 2, Analog Output</td><td>Module Name: M123_CLX_R01_S02_AO</td></tr> <tr> <td>Mixer123 Controller, Remote chassis #2, Slot 5, Discrete Input</td><td>Module Name: M123_CLX_R02_S05_DI</td></tr> <tr> <td>Mixer123 Controller, Remote chassis #2, Slot 6, Discrete Output</td><td>Module Name: M123_CLX_R02_S06_DO</td></tr> <tr> <td>Mixer123 Controller, Local chassis, Slot 5, Remote I/O</td><td>Module Name: M123_CLX_R02_S06_RIO</td></tr> </table>	Analog input	AI	Analog output	AO	Discrete input	DI	Discrete output	DO	Analog input/output combination	AIO	Discrete input/output combination	DIO	Analog/discrete input/output combination	ADIO	Serial data	SIO	Motion data	MIO	DeviceNet® data	DNET	EtherNet/IP™ data	ENET	ControlNet®	CNET	Mixer123 Controller, Local chassis, Slot 4, Analog Output	Module Name: M123_CLX_L00_S04_AO	Mixer123 Controller, Local chassis, Slot 12, Discrete Output	Module Name: M123_CLX_L00_S12_DO	Mixer123 Controller, Remote chassis #1, Slot 1, Analog Input	Module Name: M123_CLX_R01_S01_AI	Mixer123 Controller, Remote chassis #1, Slot 2, Analog Output	Module Name: M123_CLX_R01_S02_AO	Mixer123 Controller, Remote chassis #2, Slot 5, Discrete Input	Module Name: M123_CLX_R02_S05_DI	Mixer123 Controller, Remote chassis #2, Slot 6, Discrete Output	Module Name: M123_CLX_R02_S06_DO	Mixer123 Controller, Local chassis, Slot 5, Remote I/O	Module Name: M123_CLX_R02_S06_RIO
Analog input	AI																																						
Analog output	AO																																						
Discrete input	DI																																						
Discrete output	DO																																						
Analog input/output combination	AIO																																						
Discrete input/output combination	DIO																																						
Analog/discrete input/output combination	ADIO																																						
Serial data	SIO																																						
Motion data	MIO																																						
DeviceNet® data	DNET																																						
EtherNet/IP™ data	ENET																																						
ControlNet®	CNET																																						
Mixer123 Controller, Local chassis, Slot 4, Analog Output	Module Name: M123_CLX_L00_S04_AO																																						
Mixer123 Controller, Local chassis, Slot 12, Discrete Output	Module Name: M123_CLX_L00_S12_DO																																						
Mixer123 Controller, Remote chassis #1, Slot 1, Analog Input	Module Name: M123_CLX_R01_S01_AI																																						
Mixer123 Controller, Remote chassis #1, Slot 2, Analog Output	Module Name: M123_CLX_R01_S02_AO																																						
Mixer123 Controller, Remote chassis #2, Slot 5, Discrete Input	Module Name: M123_CLX_R02_S05_DI																																						
Mixer123 Controller, Remote chassis #2, Slot 6, Discrete Output	Module Name: M123_CLX_R02_S06_DO																																						
Mixer123 Controller, Local chassis, Slot 5, Remote I/O	Module Name: M123_CLX_R02_S06_RIO																																						

Parameter Name Prefixes

Programming structures, such as Add-On Instructions and programs support parameters for passing values. The convention for prefixes is to abbreviate the function of the parameter to three letters and an underscore, followed by additional text to clarify the specific function.

Parameter Function	Prefix	Description
Command	Cmd_	Designates a command input, either from the operator via the HMI or from the program. Examples: <ul style="list-style-type: none"> Cmd_Reset: Clear faults and reset the process Cmd_JogServo: Jog a servo axis Cmd_FillTank: Fill a tank with a liquid
Configuration	Cfg_	Designates a configuration value for the structure. Enter from the HMI or as part of a recipe. Examples: <ul style="list-style-type: none"> Cfg_JogDirection: Selects the direction a servo jogs: 0=Positive, 1=Negative Cfg_BulkFill: Selects the fill rate to use: 0=Slow Rate, 1=Fast Rate Cfg_UserUnits: Selects the measure of volume to use: 0=mm, 1=m, 2=gal Cfg_EnableInterlocks: Enable interlock functionality Cfg_EnablePermissive: Enable permissive functionality
Status	Sts_	Status of the process within the structure. Examples: <ul style="list-style-type: none"> Sts_Alarm: An alarm condition (such as a HI/LOW alarm) exists within the process Sts_ER: An error with an instruction execution within the process has been detected Sts_IndexComplete: The servo index move within the process has completed Sts_FillInProcess: The tank filling process is underway
Error	Err_	If the Sts_ER bit is on, the Err_ parameter indicates the actual error. This can be either a bit level or value level indication. <ul style="list-style-type: none"> Bit level error recording supports multiple errors simultaneously, but can require a large number of indicators to support all error states. Value-based error annunciation supports a large quantity of errors within one indicator. However, this approach requires that errors are annunciated one at a time. Examples: <ul style="list-style-type: none"> Err_Value: A nonzero value indicates an error condition Err_PCamCalcFault: Indicates that an error has occurred in an MCCP
Alarm	Alm_	If the Sts_Alm bit is on, the Alm_ parameter indicates which alarm is occurring. This can be either a bit-level or value-level indication. <ul style="list-style-type: none"> Bit-level alarming supports multiple alarms simultaneously, but can require a large number of indicators to support all alarm states. Value-based alarm annunciation supports a large quantity of alarms within one indicator. However, this approach requires alarms to be annunciated one at a time. Examples: <ul style="list-style-type: none"> Alm_Value: A nonzero value indicates an alarm condition Alm_TankHI: Indicates that a HI level condition has been detected within a tank
Input	Inp_	Real-time data used to drive the process. Designates a connection either to a real input point, a control device, or to data received from other processes. Examples: <ul style="list-style-type: none"> Inp_ServoPosition: Variable providing the input value for a position of a servo Inp_ServoRegistrationPosition: Input of the registration position of the servo Inp_InterlockOK: Input indicating external interlocks are met Inp_TankLevel: Variable providing the analog input for a tank level Inp_TankLevelFillRate
Output	Out_	Real-time data driven from the process. Designates a connection to a real output point, a control device, or to data sent to other processes. Examples: <ul style="list-style-type: none"> Out_GlueGun1: Output signal to turn of Glue Gun 1 Out_ServoCorrectionDistance: Output of a servo registration correction distance Out_OverflowValve: Output signal to open the Overflow Valve Out_TankLevelError: Output of a difference between target and actual fill level of a tank
Reference	Ref_	Complex data structures that combine input and output data. These structures pass data into a structure, where some process is performed. The results are then loaded back into the structure to be passed out of Add-On Instruction for use elsewhere. Example: Ref_PositionCamRecovery: Provides the data set for calculating a Position Cam with all offsets factored in, and the resulting Position Cam Profile to run in an MAPC instruction

Parameter Function	Prefix	Description
Parameter	Par_	Variables that are received from an external source that can be internal or external to the program. Examples: <ul style="list-style-type: none"> Par_MachineSpeed: Provides a machine's running speed Par_TargetFillLevel: Provides a tank's target fill level
Set point	Set_	Variables received from an operator or HMI and are not part of an external source. Examples: <ul style="list-style-type: none"> Set_MachineMaxSpeed: Provides the setting for a machine's maximum permissible speed Set_TankHILevel: Provides the setting for a tank's HI alarm limit
Value	Val_	Designates a value that might not be the primary output of the structure.
Report	Rpt_	Designates a value that is typically used for reporting.
Information	Inf_	Non-functional data such as a revision level or name for displaying a faceplate.
Ready	Rdy_	Command-ready bits that are typically Booleans calculated inside the control routines to reflect whether the routine let states change commands. Used with HMI faceplates to enable or disable command buttons.
Program Command (optional)	PCmd_	Command input for commands typically issued by the application program. Examples: <ul style="list-style-type: none"> PCmd_ProgReq - Request for Program Mode made by the application (as opposed to Cmd_ProgProgReq) PCmd_AutoReq - Request for Auto Mode made by the application (as opposed to Cmd_ProgAutoReq)
Operator Command (optional)	OCmd_	Command input for commands typically issued by the operator via the HMI. Examples: <ul style="list-style-type: none"> OCmd_ProgReq - Request for Program Mode made by the operator (as opposed to Cmd_OperProgReq) OCmd_AutoReq - Request for Auto Mode made by the operator (as opposed to Cmd_OperAutoReq)

Guidelines for Subroutines

Follow these parameter guidelines for subroutines.

Guideline	Description
Input and Return parameters depend on the subroutine logic.	If the subroutine needs to know the previous state of any Return parameters (the values are used elsewhere in the project), these values should also be Input parameters: <ul style="list-style-type: none"> If the subroutine contains latch/unlatch logic (holding circuits), intended outputs of the subroutine should be passed into and returned from the subroutine. If the subroutine does not contain latch/unlatch logic, intended outputs of the subroutine only need to be returned from the subroutine.
Pass complete timers in and out of subroutines.	If a subroutine needs a timer, pass the complete timer tag to the subroutine as an input and return the complete timer tag as an output. Store the timer in a buffer tag outside of the subroutine.
Create a user-defined tag to pass large numbers Input and Output parameters	Create and pass a UDT if you have several Input and Output parameters to save on execution time. The more parameters that you pass, the fewer nested JSRs you can perform.
Data types must match	For each parameter in an SBR or RET instruction, use the same data type (including any array dimensions) as the corresponding parameter in the JSR instruction. Using different data types can produce unexpected results.

Guidelines for User-defined Data Types

A UDT lets you organize or group data logically, so that all of the data associated with a device (such as a pressure transmitter or variable-frequency drive) can be grouped.

- You can mix data types, such as real or floating point values, counters, timers, arrays, Booleans, and other UDTs, within one UDT.
- You can copy a UDT from one project to another, and even from one Logix controller type to another.
- A UDT is self-documenting based on the tag names you assign, and provides a logical representation of parts or subsystems.

Naming Conventions for User-Defined Data Types

Element	Description
Prefix_	UDT_
UDT name	Function or purpose of the UDT

Examples:

Inventory tracking tag	UDT_InventoryTracking
Clean-in-place system	UDT_CIP
Two-state valve control module in control module	UDT_CMV2S
Water addition in equipment module	UDT_EM

UDT Member Order

The order in which elements are listed in the UDT can have a significant impact on memory use if several BOOL, INT, or SINT elements are defined. Memory is allocated in 4-byte (32-bit) increments, and every DINT, REAL, STRING, or sub-UDT element always start at the beginning of a 4-byte boundary.

For example, if the first element defined is a BOOL, it uses the first 4 bytes allocated to the UDT. Other BOOLs can be assigned immediately following without consuming any more memory, until the first 4 bytes are consumed. However, if the next element is a DINT, the DINT element allocates another 4 bytes even though the BOOL occupies only a single bit in the first 4 bytes. So for this example, the 31 bits of memory between the BOOL and the start of the DINT are allocated but are not accessible.

- UDT memory is allocated in 4-byte increments.
- Elements that occupy 4 bytes or more always start at a 4-byte boundary. These include DINT, REAL, STRING, any UDT, or any other complex data structure.
- Elements of smaller data types (BOOL, SINT, or INT) start on the next byte boundary that matches its size, so that all the data types in the UDT are fully contained in their respective 4-byte increments. For example, INT elements start on 2-byte boundaries, SINT elements follow at the next byte, and BOOL elements in succession occupy consecutive bits within a byte.

In the following example, the UDT on the left, UDT_Tank, has members arranged by function without regard for memory usage. This makes sense in the context of implementation, because members toward the top are ordinarily used in the software code.

However, the disjointed listing of data types in UDT_Tank consumes 25% more memory than the example UDT on the right, UDT_TankPacked. In UDT_TankPacked, the BOOL members are grouped according to their functionality, with the input BOOLs grouped at the top and the output BOOLs grouped at the bottom. As a result, the data type size is reduced from 80 bytes to 64 bytes.

Name:

UDT_Tank

Description:

Alternating BOOL elements with larger elements

Members:

Data Type Size: 80 byte(s)

Name	Data Type	Style	Description
Cmd_PumpStart	BOOL	Decimal	
Cmd_PumpStop	BOOL	Decimal	
Set_PumpSpeed	REAL	Float	
Set_Level	REAL	Float	
Cmd_AutoReq	BOOL	Decimal	
Cmd_ManualReq	BOOL	Decimal	
Inp_FillRate	REAL	Float	
Inp_DischargeRate	REAL	Float	
Alm_HighLevel	BOOL	Decimal	
Ctg_HighLevel	REAL	Float	
Alm_LowLevel	BOOL	Decimal	
Ctg_LowLevel	REAL	Float	
Out_FillValve	BOOL	Decimal	
Out_DischargeValve	BOOL	Decimal	
Out_Pump	BOOL	Decimal	
Out_PumpSpeed	REAL	Float	
Stz_AutoMode	BOOL	Decimal	
Stz_PumpRunning	BOOL	Decimal	
Val_Level	REAL	Float	
Tmr_Fault	TIMER		
Tmr_Delay	TIMER		

Name:

UDT_TankPacked

Description:

Packing BOOL elements into the 1st 4 byte chunk.

Members:

Data Type Size: 64 byte(s)

Name	Data Type	Style	Description
Cmd_PumpStart	BOOL	Decimal	
Cmd_PumpStop	BOOL	Decimal	
Cmd_AutoReq	BOOL	Decimal	
Cmd_ManualReq	BOOL	Decimal	
Set_PumpSpeed	REAL	Float	
Set_Level	REAL	Float	
Inp_FillRate	REAL	Float	
Inp_DischargeRate	REAL	Float	
Ctg_HighLevel	REAL	Float	
Ctg_LowLevel	REAL	Float	
Val_Level	REAL	Float	
Out_PumpSpeed	REAL	Float	
Out_Pump	BOOL	Decimal	
Out_FillValve	BOOL	Decimal	
Out_DischargeValve	BOOL	Decimal	
Stz_AutoMode	BOOL	Decimal	
Stz_PumpRunning	BOOL	Decimal	
Alm_HighLevel	BOOL	Decimal	
Alm_LowLevel	BOOL	Decimal	
Tmr_Fault	TIMER		
Tmr_Delay	TIMER		

For 64 bit data types such as LINT, ULINT, and LREAL, it is necessary for proper operation that these be aligned on 8 byte boundaries.

The following two examples show how this alignment is accomplished within UDTs.

Data Type	Byte Offset or Size
BOOL	0
PAD	7
LINT	8

Data Type	Byte Offset or Size
DINT	0
PAD	4
LREAL	8

Guidelines for Add-On Instructions

An Add-On Instruction encapsulates commonly used functions or device controls. It is not intended for use as a high-level hierarchical design tool. Once an Add-On Instruction is defined in a project, it behaves similarly to the built-in instructions that are already available in the programming software. The Add-On Instruction appears on the instruction toolbar and in the instruction browser.

Guideline	Description
Create Add-On Instructions in relay ladder, function block diagram, or structured text languages.	<p>Supports all Add-On Instructions and most built-in instructions. Excludes JSR/SBR/RET, JXR, FOR/BRK (relay ladder), SFR, SFP, SAR, IOT, and EVENT instructions.</p> <p>GSV/SSV instructions in an Add-On Instruction cannot reference the Module, Message, Axis, Motion Group, or Coordinate System class names.</p> <p>Add-On Instructions support function block, relay ladder, and structured text programming languages. Each of the Add-On Instruction logic areas can be any language. For example, the main logic can be function block and the prescan logic can be relay ladder.</p> <p>You can create safety Add-On Instructions in a safety task.</p>
<p>An Add-On Instruction supports parameters:</p> <ul style="list-style-type: none"> • Input (copied in) • Output (copied out) • InOut (passed by reference) 	<ul style="list-style-type: none"> • For Version 24 and earlier, you are limited to 512 total parameters: Input parameter + Output parameter + local tags (no limit on the number of InOut parameters) • For Version 28 and later, you are limited to 40 InOut parameters, but no limit on the number of Input or Output parameters) • 2 MB maximum data instance (parameters and locals) • Alarm, axis, axis group, coordinate system, message, motion group, and produced/consumed tags must exist at the program or controller scope and passed as an InOut parameter • Can include references to controller-scoped tags, program-scoped tags, and immediate values. • Input and Output parameters are limited to atomic (BOOL, SINT, INT, DINT, REAL) data types. Use the InOut parameter for LINT, user-defined, and structure data types. • DINT data types provide optimal execution. • Default values of parameters and local tags are used to initialize the data structure when a tag is created of the instruction's data type. When an existing parameter or local tag's default value is modified, the existing tag instances for that instruction are not updated. When a parameter or local tag is added to the instruction definition, the tag's default value is used in the existing tags.
Create and modify offline only.	<p>Online operation supports monitoring.</p> <p>Modifications to Add-On Instructions are made offline. Make changes once to the Add-On Instruction definition to affect all instances.</p>

Guideline	Description
An Add-On Instruction executes like a routine.	<p>A task with a higher execution priority can interrupt an Add-On Instruction. Use a UID/UIE instruction pair to make sure an Add-On Instruction's execution is not interrupted by a higher priority task.</p> <p>If you have many parameters or specialized options, consider multiple Add-On Instructions</p> <p>Calling many Add-On Instructions impacts scan time</p>
The code within an Add-On Instruction can access data that is specified only via parameters or defined as local.	Copy the local data to a parameter if you want to programmatically access it outside of an Add-On Instruction.
Use optional Scan mode logic to set up, initialize, or reset the Add-On Instruction code.	<p>An Add-On Instruction can have logic along with the main logic for the instruction.</p> <ul style="list-style-type: none"> • Prescan logic executes on controller startup. • Postscan logic executes on SFC Automatic reset. • EnableInFalse logic executes when rung condition is false.
Apply code signatures to Add-On Instructions for revision control.	<p>Add-On Instructions can be sealed with a code signature. Use the code signature for revision control and to identify any changes. For safety controllers, the signature can be used to get TÜV certification for a safety Add-On Instruction. For more information, see the Logix 5000 Controllers Add-On Instructions Programming Manual, publication 1756-PM010.</p>

Add-On Instruction Design Concepts

To be sure that specific data is passed into or out of the Add-On Instruction, use a required parameter. A required parameter must be passed as an argument in order for a call to the instruction for verification. To pass a required parameter in ladder diagrams and in structured text, specify an argument tag for the parameter.

- In a function block diagram, required Input parameters and Output parameters must be wired.
- In a ladder diagram, InOut parameters must have an argument tag.
- If a required parameter lacks an associated argument, the routine that contains the call to the Add-On Instruction does not verify.

Naming Conventions for Add-On Instructions

Component Name	Recommendations				
Add-On Instruction	<p>Start with the application name. Add a variant name, if applicable. Capitalize the first letter in all words in the name. Example:</p> <table border="1"> <tr> <td>PCam profile display</td><td>PCamProfileDisplay</td></tr> </table> <p>Suffix with underscore A0I, if space permits. Example:</p> <table border="1"> <tr> <td>PCam profile display</td><td>PCamProfileDisplay_A0I</td></tr> </table>	PCam profile display	PCamProfileDisplay	PCam profile display	PCamProfileDisplay_A0I
PCam profile display	PCamProfileDisplay				
PCam profile display	PCamProfileDisplay_A0I				

Comparison of Subroutines and Add-On Instructions

Comparison	Subroutine	Add-On Instructions
Accessibility	Within program (multiple copies)	Anywhere in controller (single copy)
Parameters	Pass by value	Pass by value or reference via InOut
Numeric parameters	No conversion, you must manage	Automatic data type conversion for Input and Output parameters InOut parameters must match declared type exactly
Parameters data types	Atomic, arrays, structures	<ul style="list-style-type: none"> • Atomic data types as In or Out parameters • LINT, user-defined, and structure data types as InOut parameters
Parameter checking	None, you must manage	Verification checks
Data encapsulation	All data at program or controller scope (accessible to anything)	Local data is isolated (only accessible within instruction)
Monitor/debug	Logic that is animated with mixed data from multiple calls	Logic that is animated with data from one calling instance
Supported programming languages	FBD, LD, SFC, ST	FBD, LD, ST
Callable from	FBD, LD, SFC, ST	FBD, LD, SFC, ST
Protection	Locked and View Only	Locked and View Only
Documentation	Routine, rung, textbox, line	Instruction description, revision information, vendor, rung, textbox, line, extended help
Execution performance	<ul style="list-style-type: none"> • JSR/SBR/RTN add overhead • All data is copied 	<ul style="list-style-type: none"> • Call is more efficient • InOut passed by reference

Comparison	Subroutine	Add-On Instructions
Memory use	Compact	<ul style="list-style-type: none"> • Call requires more memory • All references need additional memory
Edit	Both code and data can be modified offline and online in a running controller	<p>Code modifications are limited to offline in the project file and require a new download</p> <p>Data values associated can be modified online and offline</p>
Import/export	<p>All routines are imported/exported in the full project .LSK file (protected routines can be excluded or encrypted)</p> <p>Individual LD rungs and references and tags/UDTs can be imported/exported via the .LSX file</p>	<p>All Add-On Instructions are imported/exported in the full project .LSK file (protected instructions can be excluded or encrypted)</p> <p>Individual Add-On Instruction definitions and code are imported/exported via the .LSX file</p>

Comparison of Partial Import/Export and Add-On Instructions

Comparison	Partial Import/Export	Add-On Instructions
Logic	Any program, equipment phase, routine, Add-On Instruction, or user-defined data type in the project can be imported/exported via .LSX file.	Create once (single copy) and use anywhere in the same controller project.
Controller accessibility	<p>Import on-line with a running controller:</p> <ul style="list-style-type: none"> • Add programs, routines, and Add-On Instructions • Existing programs and routines can be replaced • Create tags and UDTs • Name collisions are detected automatically and you are prompted to rename or bind to existing components • The data values in the controller are maintained and new tags have their values initialized from the import file 	<p>Existing Add-On Instructions can only be edited offline.</p> <p>New Add-On Instructions can be created online or offline.</p>
Logic checking	You resolve conflicts on import.	The software verifies the components that you add to Add-On Instruction as you create it.
Data	<p>Editing member definitions of an Add-On Instruction maintains the values that are assigned to the parameters when:</p> <ul style="list-style-type: none"> • Inserting, adding, or deleting members • Rearranging (moving) members • Renaming members • Changing the data types of members <p>Values for members that are both renamed and moved in the same operation are not to be maintained.</p>	Local data is isolated (only accessible within the instruction).

Guidelines for Program Parameters

Program parameters define a data interface for programs to facilitate data sharing. Data sharing between programs can be achieved either through pre-defined connections between parameters or directly through a special notation. Unlike local tags, all program parameters are publicly accessible outside of the program. Additionally, HMI external access can be specified on individual basis for each parameter.

Standard (non-Safety) parameters can be created, edited, and deleted while online with the controller. The following exceptions apply:

- Parameters cannot be deleted while online if they are connected/bound to other parameters, or if the control logic references them.
- InOut parameters cannot be deleted while online
- InOut bindings can only be changed online through a Partial Import Online (PIO) operation

A safety parameter cannot be connected with or bound to a standard parameter or controller-scoped tag. A safety connection cannot be created, modified, or deleted in a safety-locked project. Input, Output, and Public parameters support the External Access attribute. InOut parameters do not.

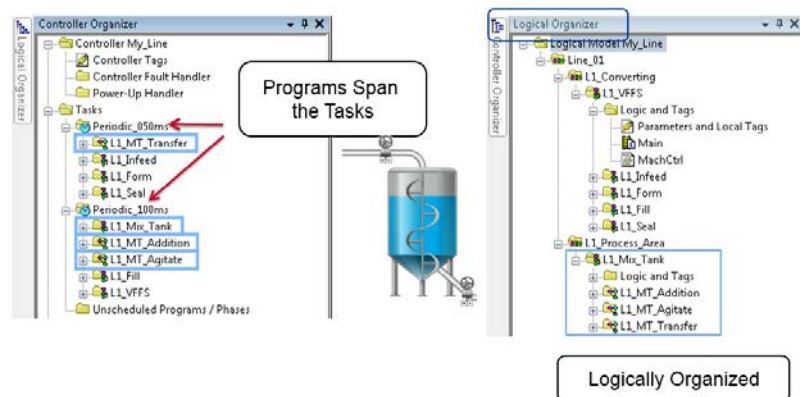
Program Parameter	Description
Input	<ul style="list-style-type: none"> • Input parameters (including members) can only support ONE connection. Only one source can be delivering the value to the input parameter. • Input Parameter values are refreshed before each scan of a program. The values do not change during the logic execution so you do not need to write code to buffer inputs. • A program can write to its own input parameters. • Data values for Output parameters that are connected to controller scope tags or Public parameters are copied after the scan of a program. In a project with multiple tasks, the data copy for a parameter that is of type BOOL, SINT, INT, DINT, LINT, or REAL will not be interrupted. A task switch can interrupt the data copy from an Output parameter to a controller scope tag or Public parameter, or any other predefined or user-defined data type.
Output	<ul style="list-style-type: none"> • Output parameters (including members) can support multiple connections. For example, lets assume you have a BOOL input parameter in Program A and Program B named Input1a and Input1b. You can connect an output parameter in Program C to Input1a AND Input1b. As stated earlier, this is often referred to as fanning. • Output Parameter values are refreshed AFTER each scan of a program. Updated output parameter values are NOT available to the parameters connected to that output parameter until the program execution is complete. • Output parameters that are connected to Public parameters or controller scope tags are copied (pushed) at the end of the program execution. • An Output parameter can ONLY be connected to an InOut parameter if both the Output and InOut parameters are configured as Constants.
InOut	<ul style="list-style-type: none"> • InOut parameters can only support ONE connection. You cannot configure connections to any member of an InOut parameter. • InOut parameters are passed by REFERENCE, which means they simply point to the base tag. In other words, when an InOut parameter is used in logic, the current value of the parameter that is connected to the InOut Parameter is used. • An InOut parameter can ONLY be connected to an Output parameter if both the Output and InOut parameters are configured as Constants. See the tool tip for Output Parameters for a more detailed explanation. • InOut parameters CANNOT be changed online, unless using the Partial Import Online (PIO).
Public	<ul style="list-style-type: none"> • Public parameters can support MULTIPLE connections. You can configure connections to the base Public parameter or any member of a Public parameter. This includes User-Defined Structures. • Public parameters are updated when the source is updated. In other words, when a Public parameter value updates, it is immediately available to any higher priority tasks that are connected to that parameter. • Public parameters can be aliased to Controller Scope Tags. If this functionality is desired, remember that the alias update is asynchronous to program execution. The public parameter contains the real-time value of the controller scope tag.

Comparison of Program Parameters and Add-On Instructions

Comparison	Program Parameters	Add-On Instructions
Accessibility	Within program (multiple copies)	Anywhere in controller (single copy)
Parameters	Input / Output (pass by value), InOut (pass by reference), Public (pass by value)	Input / Output (pass by value), InOut (pass by reference)
Numeric parameters	<ul style="list-style-type: none"> Automatic data type conversion for Input and Output parameters InOut parameters must match declared type exactly 	<ul style="list-style-type: none"> Automatic data type conversion for Input and Output parameters InOut parameters must match declared type exactly
Parameters data types	Atomic, strings, arrays, structures	<ul style="list-style-type: none"> Atomic data types as In or Out parameters LINT, user-defined, and structure data types as InOut parameters
Parameter checking	None, user must manage	Verification checks
Data encapsulation	All data at program or controller scope (accessible to anything). Programs can talk directly and exchange data between them. Local tags remain private to the Program. Cannot access Local Tags, only the parameters.	Local data is isolated (only accessible within instruction)
Monitor/debug	Online editable.	Logic that is animated with data from one calling instance
Supported programming languages	FBD, LD, SFC, ST	FBD, LD, ST
Callable from	FBD, LD, SFC, ST	FBD, LD, SFC, ST
Protection	—	Locked and View Only
Documentation	—	Instruction description, revision information, vendor, rung, textbox, line, extended help
Execution performance	<ul style="list-style-type: none"> Programs can talk directly and exchange data between them. InOut passed by reference 	<ul style="list-style-type: none"> Call is more efficient InOut passed by reference
Memory use	Compact. One Public parameters can be connected or bound to multiple Input, Output or InOut parameters to form a shared memory space.	<ul style="list-style-type: none"> Call requires more memory All references need additional memory
Edit	Online editable, and supports sub-element connections. Copy / Paste Programs without disturbing parameter configuration.	Code modifications are limited to offline in the project file and require a new download Data values associated can be modified online and offline

Compare Controller Organizer and Logical Organizer

The Controller Organizer presents program logic how the controller executes the logic. The Logical Organizer is configurable to present program logic how the user views the system.



Structure Logic According to Standards

The ANSI/ISA-88.01-1995 (R2006) standard is the most recognized and broadly adopted standard for modular equipment control. Complementing ISA-88.01 is the ISA-TR88.00.02 technical report, which is also known as PackML. The ISA-TR88.00.02 technical report comes from the Organization for Machine Automation and Control (OMAC). The OMAC provides examples of how to apply ISA-88.01 in discrete manufacturing segments.

ISA refers to the International Society of Automation, an ANSI recognized Standards Development Organization (SDO).

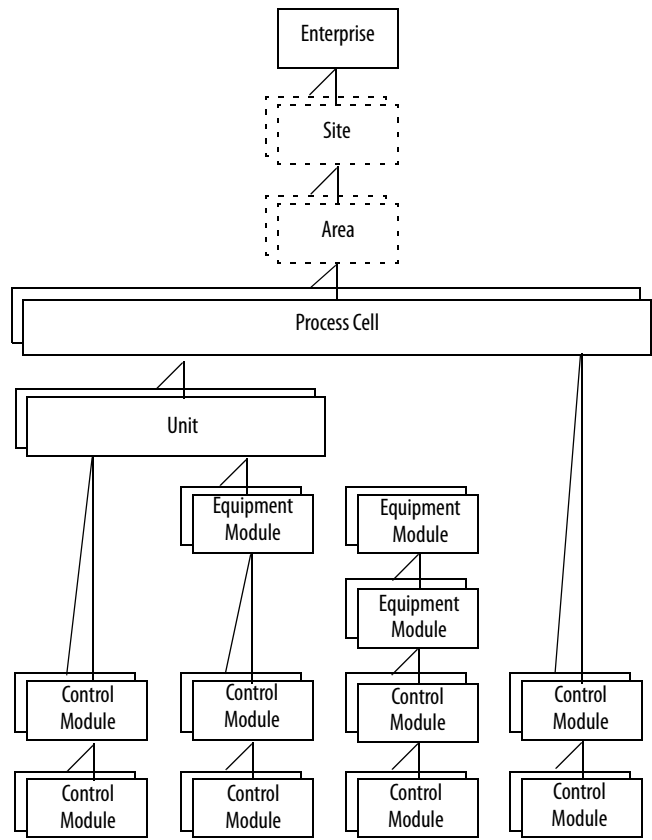
- ISA-88 refers to a specific set of ISA standards, of which ISA88.01 is a subset.
- SP88 refers to the ISA working member group responsible for the creation and publication of the ISA-88 standards.

ISA-88 provides these models to define and understand the automation control requirements for manufacturing plants.

Model	Description
Physical	The physical model (also known as the equipment model) describes a hierarchical organization of equipment and the basic control capabilities that are associated with that organization. The physical model is a representation of the equipment in the plant that makes the product.
Procedural	The procedural model describes a multi-tiered, hierarchical model that defines the process capability and automation control in relation to the Physical Model to perform a task. The procedural model is a representation of how to use the equipment (described in the physical model) to make the product.

Physical Model

The physical model (also known as the equipment model) describes a hierarchical organization of equipment and the basic control capabilities that are associated with that organization. The physical model is a representation of the equipment in the plant that is used to make the product.



The physical model defines the automation components within a given environment, and determines modular areas and component interactions.

The physical model shows that an enterprise can contain multiple sites, and a site can contain multiple areas, down to the equipment modules and control modules that carry out the manufacturing process

Physical Model Component	Description
Enterprise	The company that owns the facilities.
Site	The location of one facility.
Process cell	A collection of one or more units that are linked together to perform a task or multiple tasks of the process for one or more products in a defined sequence. A process cell contains the units, equipment modules, and control modules that are needed to make one or more batches.

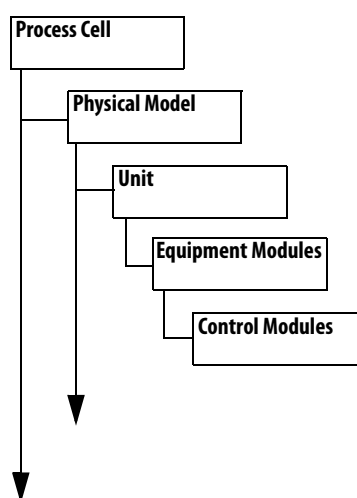
Physical Model Component	Description
Unit (or machine)	A collection of related equipment modules and control modules that execute one or more processing activities. The unit corresponds to the logical grouping of mechanical and electrical assemblies that historically has been called a machine. The term unit can apply to either a single-function machine or a multi-functional machine.
Equipment module	<p>A functional group of control modules, equipment modules, or both that execute a finite number of activities. The primary purpose of control in an equipment module is to coordinate the functions of other equipment modules and lower-level control modules. A process cell, unit, operator, or another equipment module can command an equipment module.</p> <p>An equipment module can be part of A unit, or a standalone equipment group can include an equipment module in a process cell. If engineered as a standalone equipment grouping, an equipment module can be an exclusive-use resource or a shared-use resource. The equipment module combines all necessary physical processing and control equipment that is required to perform the manufacturing process. The finite tasks that an equipment module is designed to carry out defines the scope of the equipment module.</p> <p>The terms control module and equipment module apply to the physical equipment as well as to the equipment entity. The following are examples of equipment modules.</p> <ul style="list-style-type: none"> • A valve matrix used for material transfer between units (shared resource of process cell) • A level control for a tank (equipment module within a specific unit) • A vertical form-fill-seal machine's 'sealing jaws control' (equipment module within a discrete unit)
Control module	<p>Control module: A regulating device, state-oriented device, or combination thereof (typically, a collection of sensors, actuators, and other control modules) that is operated as a single device. Control that is normally found at this level directly manipulates actuators and other control modules. A control module can direct commands to other control modules, or to actuators that have been configured as part of the control module. Control of the process is affected through the equipment-specific manipulation of control modules and actuators. The control module is the lowest level grouping of equipment in the physical model that can carry out basic control.</p> <p>The following are examples of control modules.</p> <ul style="list-style-type: none"> • An individual sensor or actuator • A state-oriented device that consists of an on/off automatic block valve with position feedback switches and that is operated via the set point of the device • A header that contains several on/off automatic block valves that coordinates the valves to direct flow to one or several destinations based on the set point directed to the header CM • A servo-controlled electronic gear or cam function (that is, a discrete unit), including its interlock and permissives. <p>The following are typical control modules in a programming library:.</p> <ul style="list-style-type: none"> • Analog output • Analog input with scaling and alarms • Reversing motor • Variable speed drive • Solenoid-operated 2-state valve • Motor-operated 2-state valve • PID with standard modes and deviation alarms

Separate a Process Unit into Equipment Modules and Control Modules

To create a modular control program, start with one of these approaches.

- Identify the control modules in the process. Then group the control modules into equipment modules to be supervised and coordinated by procedural controls.
- Determine the units (typically vessels containing a single batch at a time). Then determine the equipment modules (such as ingredient addition equipment, agitating equipment, thermal jacket temperature control equipment, and transfer out equipment) by reviewing the related equipment, piping, and instrumentation on a process and instrumentation diagram (P&ID). Then determine the control modules that are related to the equipment states that must be controlled (such as motors, valves, or other process control loops).

One way to organize application logic in the Controller Organizer is to create a separate folder for each unit.



Physical Model Naming Conventions

Component Name	Recommendations		
Site	Short, preferably single character abbreviation, all upper case, of the formal name of the site Example: <table border="1"> <tr> <td>Site: My Site</td><td>Site Name: M</td></tr> </table>	Site: My Site	Site Name: M
Site: My Site	Site Name: M		
Area	Building number prefixed by a site name, but with no separating underscore Example: <table border="1"> <tr> <td>Area: Manufacturing Building (102)</td><td>Area Name: M102</td></tr> </table>	Area: Manufacturing Building (102)	Area Name: M102
Area: Manufacturing Building (102)	Area Name: M102		

Component Name	Recommendations																
Cell	<p>Two- to three-character abbreviation of the formal name of the cell Prefix with an area name and a single underscore (optional) Example:</p> <table> <tr> <td>Cell: Mixing</td><td>Cell Name: A102_MIX_123</td></tr> </table>	Cell: Mixing	Cell Name: A102_MIX_123														
Cell: Mixing	Cell Name: A102_MIX_123																
Unit	<p>In a unit class, prefix the unit name with UN and an underscore Example:</p> <table> <tr> <td>Unit Class: Mix Tank</td><td>Unit Name: (Machine)</td></tr> <tr> <td>Unit Class: Packer</td><td>Unit Name: UN_Pckr</td></tr> </table> <p>In a unit instance, use the unit identification from the piping and instrumentation diagram (P&ID) All uppercase letters; use underscored instead of dashes Prefix the unit name with the area name, separated from the unit name by a single underscore Example:</p> <table> <tr> <td>Unit Instance: Mix Tank 1234</td><td>Unit Name: A102_MT1234</td></tr> <tr> <td>Machine Instance: Packaging Machine 1234</td><td>Unit Name: A102_PKR1234</td></tr> </table> <p>In a unit program instance, add the unit identifier between the area name and the unit name Example:</p> <table> <tr> <td>Unit Instance Program Name:</td><td>A102_UN02_MT1234</td></tr> <tr> <td>Machine Instance Program Name:</td><td>A102_UN02_PKR1234</td></tr> </table> <p>In a unit tag instance, prefix the tag with the unit name Example:</p> <table> <tr> <td>Unit Instance Tag Name:</td><td>MT1234</td></tr> <tr> <td>Machine Instance Tag Name:</td><td>PKR1234</td></tr> </table>	Unit Class: Mix Tank	Unit Name: (Machine)	Unit Class: Packer	Unit Name: UN_Pckr	Unit Instance: Mix Tank 1234	Unit Name: A102_MT1234	Machine Instance: Packaging Machine 1234	Unit Name: A102_PKR1234	Unit Instance Program Name:	A102_UN02_MT1234	Machine Instance Program Name:	A102_UN02_PKR1234	Unit Instance Tag Name:	MT1234	Machine Instance Tag Name:	PKR1234
Unit Class: Mix Tank	Unit Name: (Machine)																
Unit Class: Packer	Unit Name: UN_Pckr																
Unit Instance: Mix Tank 1234	Unit Name: A102_MT1234																
Machine Instance: Packaging Machine 1234	Unit Name: A102_PKR1234																
Unit Instance Program Name:	A102_UN02_MT1234																
Machine Instance Program Name:	A102_UN02_PKR1234																
Unit Instance Tag Name:	MT1234																
Machine Instance Tag Name:	PKR1234																
Equipment module	<p>Area name, underscore, letters EM, underscore, function abbreviation All uppercase letters Example:</p> <table> <tr> <td>EM Instance: Mix Tank 1234 Vessel Agitator</td><td>EM Name: MT1234_EM_Agitate</td></tr> <tr> <td>EM Instance: Packager 1234 Bag Forming</td><td>EM Name: PKR1234_EM_BagForming</td></tr> </table>	EM Instance: Mix Tank 1234 Vessel Agitator	EM Name: MT1234_EM_Agitate	EM Instance: Packager 1234 Bag Forming	EM Name: PKR1234_EM_BagForming												
EM Instance: Mix Tank 1234 Vessel Agitator	EM Name: MT1234_EM_Agitate																
EM Instance: Packager 1234 Bag Forming	EM Name: PKR1234_EM_BagForming																
Control Module	<p>In a control module class, prefix the name with CM and an underscore Example:</p> <table> <tr> <td>CM Class: PID Control Loop</td><td>CM Name: CM_PID</td></tr> <tr> <td>CM Class: Cylinder</td><td>CM Name: CM_Cylndr</td></tr> </table> <p>In a control module instance, use all uppercase letters Area name, underscore, unit name, underscore, instrument name, loop sequence number Example:</p> <table> <tr> <td>CM Instance: Temperature Controller 01</td><td>CM Name: TC_01</td></tr> <tr> <td>CM Instance: Cylinder 01</td><td>CM Name: CY01</td></tr> </table>	CM Class: PID Control Loop	CM Name: CM_PID	CM Class: Cylinder	CM Name: CM_Cylndr	CM Instance: Temperature Controller 01	CM Name: TC_01	CM Instance: Cylinder 01	CM Name: CY01								
CM Class: PID Control Loop	CM Name: CM_PID																
CM Class: Cylinder	CM Name: CM_Cylndr																
CM Instance: Temperature Controller 01	CM Name: TC_01																
CM Instance: Cylinder 01	CM Name: CY01																

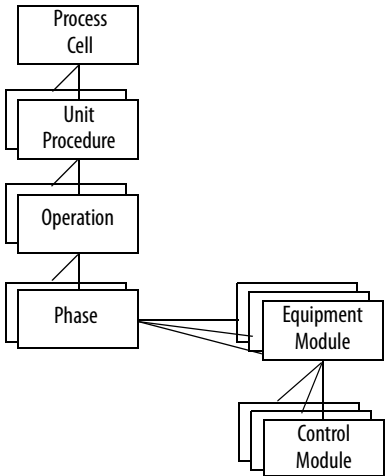
Procedural Model

The procedural model describes a multi-tiered, hierarchical model that defines the process capability and automation control in relation to the physical model to perform a task. The procedural model is a representation of **how to use** the equipment (described in the physical model) to make the product.

The procedural control that is laid out in the procedural model directs the equipment components, via the component interfaces, to perform the specific tasks out of the available capabilities, needed to produce a given product.

Procedural Model Component	Description
Procedure	The general strategy for production within a process cell. A procedure is composed of unit procedures.
Unit procedure	A production sequence. Unit procedures are composed of operations.
Operation	The single sequence necessary for the initiation, organization, and control of phases. Operations are composed of phases.
Phase	<p>The lowest level of a procedure that can accomplish an action.</p> <p>The intent of a phase is to cause or define a process-oriented action, while the set of steps in the phase is equipment-specific.</p> <ul style="list-style-type: none">• A phase can be subdivided into smaller parts.• A phase can issue one or more commands or cause one or more actions.• The execution of a phase can result in additional commands.

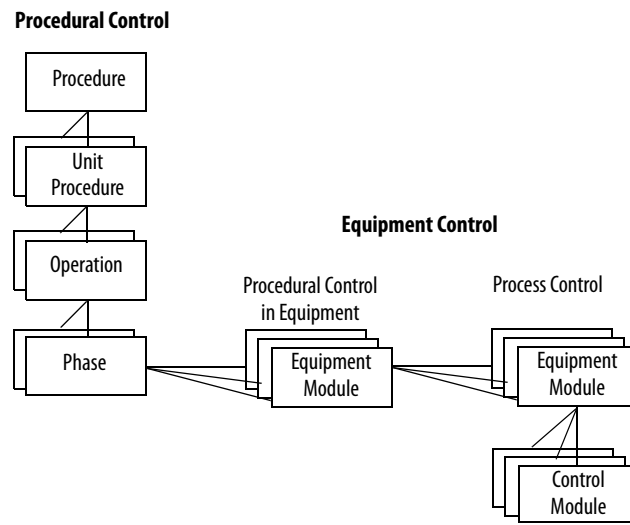
Combine the procedural model with the physical model to reflect the hierarchy of control and equipment, as well as the vertical separation between process controls and procedural controls.



Not all manufacturing processes require the procedural control to reside in the physical equipment.

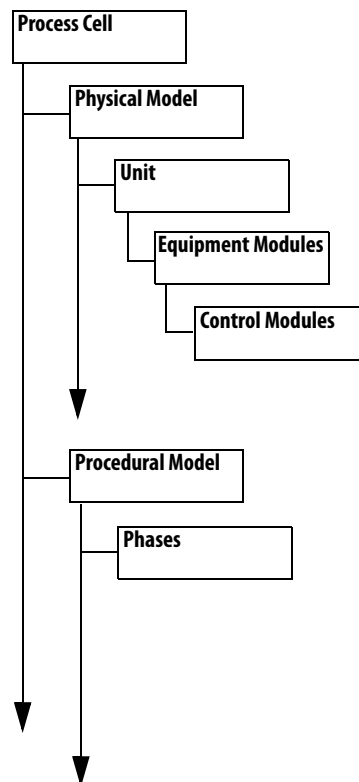
In a distributed or flexible process, procedural control can reside outside the equipment, in what is called the control recipe. Examples of this type of manufacturing are large batch systems, material handling systems, or automotive assembly systems. Use of the control recipe lets manufacturers with different automation requirements separate procedural control and process control.

The procedural control is where distributed and flexible control occurs, and where the need arises for a separation from process control.



Identify Operations and Phases

One way to organize application logic in the Controller Organizer is to add the procedural model as a separate folder of phases.



Procedural Control Modes

A control mode determines how equipment entities and procedural elements respond to commands and how they operate. The mode determines how the procedure progresses and who can affect that progression. For example, in a control module that contains basic control functions, such as an automatic block valve, the mode determines what drives the valve position and who can manipulate the position.

The ISA-88 standard defines the following modes.

Control Mode	Behavior	Command
Automatic (procedural)	Transitions within a procedure are conducted without interruption as appropriate conditions are met.	Operators can pause the progression, but can not force transitions.
Automatic (basic control)	Equipment entities are manipulated by their control algorithm.	Equipment entities cannot be manipulated directly by the operator.
Semi-Automatic (procedural)	Transitions within a procedure are conducted on manual commands as appropriate conditions are fulfilled.	Operators can pause the progression or redirect the execution to an appropriate point. Transitions cannot be forced.
Manual (procedural)	Procedural elements within a procedure are executed in the order that is specified by an operator.	Operators can pause the progression or force transitions.
Manual (basic control)	Equipment entities are not manipulated by their control algorithm.	Equipment entities can be manipulated directly by the operator.

Procedural Control States

The ISA-88 standard defines the following states.

Control State	Description
Idle	The procedural element is waiting for a Start command to cause a transition to the Running state.
Running	The procedural element is operating normally.
Complete	The procedural element has run to completion and is now waiting for a reset command that prompts a transition to Idle.
Pausing	The procedural element or equipment entity received a Pause command. This causes the procedural element to stop at the next defined safe or stable stop location in its normal Running logic. Once the procedural element has stopped, the procedural element automatically transitions from Pausing to Paused.
Paused	Once the procedural element is paused at a defined stop location, it transitions from a Pausing to Paused. The Paused state is usually used for short-term pauses. A Resume command starts a transition to the Running state, and the procedural element resumes normal operation immediately following the defined stop location.
Holding	The procedural element received a Hold command and is executing its Holding logic to put the procedural element or equipment entity into a known state. If no sequencing is required, then the procedural element or equipment entity transitions immediately to the Held state.
Held	The procedural element completed its Holding logic and proceeded to the Held state. This state is usually used for a long-term stop. The procedural element or equipment entity waits for a further command to proceed.
Restarting	The procedural element receives a Restart command while in the Held state and executes its restart logic to return to the Running state. If no sequencing is required, then the procedural element or equipment entity transitions immediately to the Running state.
Stopping	The procedural element received a Stop command and is executing its Stopping logic that facilitates a controlled normal stop. If no sequencing is required, then the procedural element or equipment entity transitions immediately to the Stopped state.

Control State	Description
Stopped	The procedural element or equipment entity completed its Stopping logic and waits for a Reset command to transition to an Idle state.
Aborting	The procedural element received an Abort command and is executing its Abort logic that is the logic that facilitates a quicker, but not necessarily controlled, abnormal stop. If no sequencing is required, then the procedural element transitions immediately to the Aborted state.
Aborted	The procedural element completed its Aborting logic and waits for a Reset command to transition to the Idle state.

Procedural Control Commands

The ISA-88 standard defines the following commands.

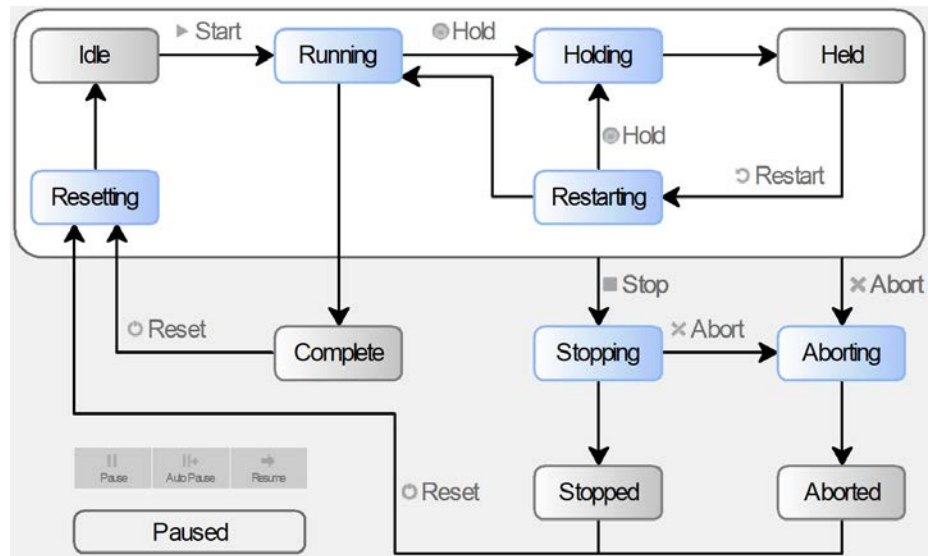
Command	Description	Valid States
Start	Orders the procedural element to begin executing the normal Running logic.	Idle
Stop	Orders the procedural element to execute the Stopping logic.	Running Pausing Paused Holding Held Restarting
Hold	Orders the procedural element to execute the Holding logic.	Running Pausing Paused Restarting
Restart	Orders the procedural element to execute the Restarting logic to safely return to the Running state.	Held
Abort	Orders the procedural element to execute the Aborting logic.	Running Pausing Paused Holding Held Restarting Stopping Stopped
Reset	Orders the procedural element to transition to an Idle state.	Complete Aborted Stopped
Pause	Orders the procedural element to pause at the next programmed pause transition within its sequencing logic and await a Resume command before proceeding.	Running
Resume	Orders a procedural element that is in the Paused state, due to either a Pause command or a Single Step mode, to resume normal operation.	Paused

Procedural Model Naming Conventions

Component Name	Recommendations				
Procedure	<p>Use all upper case letters Prefix with type of procedure (such as RP for recipe procedure), underscore, abbreviation of procedure Example:</p> <table> <tr> <td>Recipe Procedure Class: Clean</td><td>Procedure Name: RP_CLEAN</td></tr> </table> <p>For a procedure executed in a Logix controller, prefix with area name, underscore, cell name, underscore Example:</p> <table> <tr> <td>Mixing, Tank: Clean</td><td>Procedure Name: M102_MIX_RP_CLEAN</td></tr> </table>	Recipe Procedure Class: Clean	Procedure Name: RP_CLEAN	Mixing, Tank: Clean	Procedure Name: M102_MIX_RP_CLEAN
Recipe Procedure Class: Clean	Procedure Name: RP_CLEAN				
Mixing, Tank: Clean	Procedure Name: M102_MIX_RP_CLEAN				
Unit procedure	<p>Use all upper case letters Prefix with UP, underscore, abbreviation of unit procedure Example:</p> <table> <tr> <td>Unit Procedure Class: Clean</td><td>Unit Procedure Name: UP_CLEAN</td></tr> </table> <p>For a unit procedure executed in a Logix controller, prefix with area name, underscore, unit name, underscore Example:</p> <table> <tr> <td>Unit Procedure Class: Mix Tank Clean</td><td>Procedure Name: M102_TK2333_UP_CLEAN</td></tr> </table>	Unit Procedure Class: Clean	Unit Procedure Name: UP_CLEAN	Unit Procedure Class: Mix Tank Clean	Procedure Name: M102_TK2333_UP_CLEAN
Unit Procedure Class: Clean	Unit Procedure Name: UP_CLEAN				
Unit Procedure Class: Mix Tank Clean	Procedure Name: M102_TK2333_UP_CLEAN				
Operation	<p>Use all upper case letters Prefix with OP, underscore, abbreviation of operation Example:</p> <table> <tr> <td>Operation Class: Steam-In-Place</td><td>Operation Name: OP_SIP</td></tr> </table> <p>For an operation executed in a Logix controller, prefix with area name, underscore, unit name, underscore Example:</p> <table> <tr> <td>Mix Tank: Steam-In-Place Operation</td><td>Operation Name: M102_TK2333_OP_SIP</td></tr> </table>	Operation Class: Steam-In-Place	Operation Name: OP_SIP	Mix Tank: Steam-In-Place Operation	Operation Name: M102_TK2333_OP_SIP
Operation Class: Steam-In-Place	Operation Name: OP_SIP				
Mix Tank: Steam-In-Place Operation	Operation Name: M102_TK2333_OP_SIP				
Phase	<p>Use all uppercase letters Unit identifier, underscore, function abbreviation, EP, underscore, abbreviation of phase Example:</p> <table> <tr> <td>EP Instance: Mix Tank 1234 Vessel Agitator</td><td>EP Name: MT1234_EP_Agitate</td></tr> <tr> <td>EP Instance: Packager 1234 Bag Forming</td><td>EP Name: PKR1234_EP_BagForming</td></tr> </table>	EP Instance: Mix Tank 1234 Vessel Agitator	EP Name: MT1234_EP_Agitate	EP Instance: Packager 1234 Bag Forming	EP Name: PKR1234_EP_BagForming
EP Instance: Mix Tank 1234 Vessel Agitator	EP Name: MT1234_EP_Agitate				
EP Instance: Packager 1234 Bag Forming	EP Name: PKR1234_EP_BagForming				

State Model

A state model helps program the equipment in a structured way, which results in the same behavior in all equipment throughout the plant.



Command		Start	Stop	Hold	Restart	Abort	Reset	Pause	Resume
Initial State	No Command State	State Transition Matrix							
Idle		Running	Stopping			Aborting			
Running	Complete		Stopping	Holding		Aborting		Pausing	
Complete							Idle		
Pausing	Paused		Stopping	Holding		Aborting			
Paused			Stopping	Holding		Aborting			Running
Holding	Held		Stopping		Restarting	Aborting			
Held			Stopping	Holding		Aborting			
Restarting	Running		Stopping	Holding		Aborting			
Stopping	Stopped					Aborting			
Stopped						Aborting	Idle		
Aborting	Aborted								
Aborted							Idle		
Resetting	Idle		Stopping			Aborting			

Notes:

Produced and Consumed Data

The controllers support the ability to produce (broadcast) and consume (receive) system-shared tags.

For two controllers to share produced or consumed tags, both controllers must be in the same backplane or attached to the same control network. You cannot bridge produced and consumed tags over two networks.

IMPORTANT The actual number of produced and consumed tags that you can configure over ControlNet® or EtherNet/IP™ in a project depends on the connection limits of the communication module through which you produce or consume the tags.

For more information on produced and consumed tags, see the Logix 5000 Controllers Produced and Consumed Tags Programming Manual, publication [1756-PM011](#).

Guidelines for Produced and Consumed Tags

Guideline	Description
You cannot bridge produced and consumed tags over different networks.	For two controllers to share produced or consumed tags, both controllers must be attached to the same network. You can produce and consume tags over ControlNet or EtherNet/IP networks.
Create the tag at controller scope.	You can only produce and consume (share) controller-scoped tags.
Limit the size of the tag to ≤ 500 bytes.	If you produce or consume a tag over a network, the tag must be ≤ 480 bytes. Network transfers require 20 bytes of data overhead.
Combine data that goes to the same controller.	If you are producing several tags for the same controller: <ul style="list-style-type: none"> Group the data into one or more user-defined structures. This uses fewer connections than producing each tag separately. Group the data according to similar update intervals. To conserve network bandwidth, use a greater RPI for less critical data.
The following atomic data types can be directly produced or consumed: DINT UDINT LINT ULINT REAL LREAL	The listed atomic data types can be produced or consumed individually or in arrays. Other atomic data types (BOOL, SINT, USINT, INT, UINT) cannot be done individually or in arrays and are required to be part of a data structure to be produced or consumed.
Data structures can be produced or consumed	Data structures include user-defined data types (UDT), strings, add-on-defined, predefined, and module-defined data structures. Data structures meeting the size requirements can be produced or consumed.
AXIS_CIP_DRIVE	Use AXIS_CIP_DRIVE to produce and consume axis data.
The data type in the producer and the consumer must match.	The data type for a produced or consumed tag must be the same in both the producer and the consumer.
Produce tags that are based on user-defined structures to non-Logix devices.	The controller produces tags in 32-bit words. For devices that communicate in other word boundaries, such as 16-bit words, the resulting data in the target device can be misaligned. To help avoid misalignment, structure the produced data in a user-defined structure.
Use a programmatic handshake to help ensure data is exchanged.	Produced tags continually transmit based on the RPI, so it can be difficult to know when new data arrives. You can set a bit or increment a counter that is embedded in the produced tag to identify to the consumer that new data is present. You can also provide a return handshake via a reverse produced/consumed tag, so that the original producer knows that the consumer received and processed the tag.
Use a CPS instruction to buffer produced and consumed data.	Use the CPS instruction to copy the data to the outgoing tag on the producer side. Then use another CPS instruction to copy the data into a buffer tag on the consumer side. The CPS instructions provide data integrity for data structures greater than 32 bits. Important: The controller inhibits all interrupts while it executes a CPS instruction.
Use unicast EtherNet/IP communication to reduce broadcast network traffic.	To reduce bandwidth use and preserve network integrity, some facilities block multicast Ethernet packets. You can configure a produced and consumed tag to use multicast or unicast connections. Unicast connections help with the following: <ul style="list-style-type: none"> Reduce network bandwidth Simplify Ethernet switch configuration
Monitoring produced and consumed data.	Group produced and consumed tags as members in user-defined structures whose first member is a CONNECTION_STATUS type. This technique helps monitor connection status between controllers without increasing execution time, such as using a GSV instruction to detect status.
Firmware revisions	When adding the Producer controller to the I/O configuration list of the Consumer controller, the firmware revision does not have to match. However, the rack size and slot number must be correct.

Guidelines for Produced and Consumed Axis

Guideline	Description
You can configure a produced and consumed axis between controllers in a chassis or over an EtherNet/IP network.	Controllers that support motion support the AXIS_CIP_DRIVE data type for produced and consumed tags.
Use a produced and consumed axis to synchronize motion functions across multiple controllers.	Synchronize functions such as: <ul style="list-style-type: none"> • PCAM • GEAR • MDSC moves • Scheduled outputs • Registration events • Position-based interlocks (handshake)

Guidelines to Specify an RPI Rate for Produced and Consumed Tags

When configuring produced and consumed tags, you specify a requested packet interval (RPI) rate. The RPI value is the rate at which the controller attempts to communicate with the module.

Guideline	Description
Make sure that the RPI is equal to or greater than the NUT.	You use RSNetWorx™ for ControlNet software to select the network update time (NUT) and the software schedules the network connections. RSNetWorx software cannot schedule a ControlNet network if a module and/or produced/consumed tag on the network has an RPI that is faster than the network update time.
RPI of multicast tags	<ul style="list-style-type: none"> • For Studio 5000 Logix Designer® application version 24 and earlier: The smallest (fastest) consumer RPI determines the RPI for the produced tag. If multiple consumers request the same tag, the smallest (fastest) request determines the rate at which the tag is produced for all consumers. • For Studio 5000 Logix Designer application version 28 and later: the first consumer of a produce tag determines the RPI at which data is produced. All subsequent consumers must request the same RPI value as the first consumer. Otherwise, the subsequent consumers fail to connect.

Guidelines to Manage Connections for Produced and Consumed Tags

Guideline	Description
Minimize the use of produced and consumed tags.	To reduce network traffic, minimize the size of produced and consumed tags. Also, minimize the use of produced and consumed tags to high-speed, deterministic data, such as interlocks.
Use arrays or user-defined structures.	When sending multiple tags to the same controller, use an array or user-defined structure to consolidate the data. The byte limit of ≤ 500 bytes per produced and consumed tag still applies.
Configure the number of consumers accurately.	Make sure the number of consumers that are configured for a produced tag is the actual number of controllers that consumes the tag. If you set the number higher than the actual number of controllers, you unnecessarily use up connections. The default is two consumers per produced tag.
Multiple produced/consumed connections are linked.	If there are multiple produced and consumed connections between two controllers and one connection fails, all produced and consumed connections fail. Consider combining all produced and consumed data into one structure or array so that you only need one connection between the controllers.

Configure an Event Task Based on a Consumed Tag

An event task executes automatically based on a preconfigured event occurring. One such event can be the arrival of a consumed tag.

- Only one consumed tag can trigger a specific event task.
- Use an IOT instruction in the producing controller to signal the production of new data.
- When a consumed tag triggers an event task, the event task waits for all data to arrive before the event task executes.

Compare Messages and Produced/Consumed Tags

Method	Benefits	Considerations
Read/Write Message	<ul style="list-style-type: none"> • Programmatically initiated • Communication and network resources that are only used when needed • Support automatic fragmentation and reassembly of large data packets, up to as many as 32,767 elements • Some connections can be cached to improve retransmission time • Generic CIP™ message useful for third-party devices 	<ul style="list-style-type: none"> • Delay can occur if resources are not available when needed • MSG instruction and processing can impact controller scan (system overhead timeslice) • Data arrives asynchronous to program scan (use a programmatic handshake or a UID/UIE instruction pair to reduce impact, no event task support) • Can add additional messages online in Run mode.
Produced/Consumed Tag	<ul style="list-style-type: none"> • Configured once and sent automatically based on requested packet interval (RPI) • Multiple consumers can simultaneously receive the same data from a produced tag • Can trigger an event task when consumed data arrives • ControlNet resources are reserved up front • Does not affect the scan of the controller 	<ul style="list-style-type: none"> • Support limited to Logix 5000™ and PLC-5® controllers, and the 1784-KTCS I/O Linx and select third-party devices • Limited to 500 bytes over the backplane and 480 bytes over a network • Must be scheduled when using ControlNet • Data arrives asynchronous to program scan (use a programmatic handshake or CPS instruction and event tasks to synchronize) • Connection status must be obtained separately • You can configure status information for a produced/consumed tag • On an EtherNet/IP network, you can configure produced/consumed tags to use multicast or unicast connections. • Cannot create additional produced/consumed tags online in Run mode.

Data Structures

The controllers support IEC 61131-3 atomic data types, and compound data types, such as arrays, predefined structures (such as counters and timers), and user-defined structures.

Guidelines for Data Types

Follow these guidelines depending on the data type for your application.

Guideline	Description
Use DINT data types whenever possible	The controllers perform DINT (32 bit) and REAL (32 bit) math operations. DINT data types use less memory and execute faster than other data types. Use the following data type: <ul style="list-style-type: none"> DINT for most numeric values and array indexes. REAL for manipulating floating point, analog values. SINT (8 bit) and INT (16 bit) primarily in user-defined structures or when communicating with an external device that does not support DINT values.
Group BOOL values into arrays	When you use BOOL values, group them into DINT arrays to best use controller memory and to make the bits accessible via FBC or DDT instructions.

Memory	SINT	INT	DINT	REAL
Memory that is reserved for a standalone tag	4 bytes	4 bytes	4 bytes	4 bytes
Memory that is reserved for data in a user-defined structure	1 byte (8-bit aligned)	2 bytes (16-bit aligned)	4 bytes (32-bit aligned)	4 bytes (32-bit aligned)
Memory that is used to access a tag in an ADD instruction	236 bytes	260 bytes	28 bytes	44 bytes

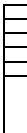
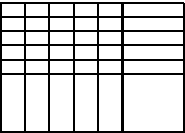
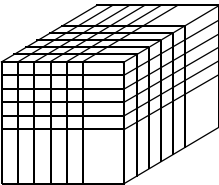
A tag uses additional memory in the controller to store the tag name and symbol, and allocate memory for data.

To manipulate SINT or INT data, the controller converts the values to DINT values, performs the programmed manipulation, and then returns the result to a SINT or INT value. This requires additional memory and execution time when compared to using DINT values for the same operation.

Arrays

An array allocates a contiguous block of memory to store a specific data type as a table of values.

- Tags support arrays in one, two, or three dimensions.
- User-defined structures can contain a single-dimension array as a member of the structure.

This array	Stores Data like	For Example				
One dimension		Tag name	Type	Dimension 0	Dimension 1	Dimension 2
		<i>one_d_array</i>	DINT[7]	7	—	—
		Total number of elements = 7 Valid subscript range DINT[a] where a=0...6				
Two dimension		Tag name	Type	Dimension 0	Dimension 1	Dimension 2
		<i>two_d_array</i>	DINT[4,5]	4	5	—
		Total number of elements = 4 * 5 = 20 Valid subscript range DINT[a,b] where a=0...3; b=0...4				
Three dimension		Tag name	Type	Dimension 0	Dimension 1	Dimension 2
		<i>three_d_array</i>	DINT[2,3,4]	2	3	4
		Total number of elements = 2 * 3 * 4 = 24 Valid subscript range DINT[a,b,c] where a=0...1; b=0...2, c=0...3				

The data type you select for an array determines how the contiguous block of memory gets used.

BOOL[96] = 12 bytes

BOOL arrays use 32-bit increments of memory

[illegible]

SINT[10] = 12 bytes of memory (2 bytes unused)

SINT arrays are padded to use any left over bytes

3	2	1	0
7	6	5	4
Unused	Unused	9	8

INT[5] = 12 bytes of memory (2 bytes unused)

INT arrays are padded to use any left over bytes

1	0
3	2
Unused	4

DINT[3] = 12 bytes and REAL[3] = 12 bytes

DINT and REAL arrays use 4-byte increments of memory

0
1
2

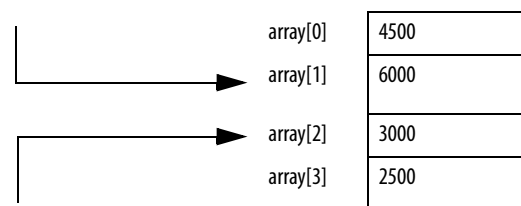
Guidelines for Arrays

Guideline	Description	
You can create arrays of most data types, except for ALARM_xxx, AXIS_xxx, COORDINATE_SYSTEM, ENERGY_xxx, HMIBC, MESSAGE, and MOTION_GROUP data types.	<p>A subscript identifies an individual element within the array. A subscript starts at 0 and extends to the number of elements minus 1 (zero based).</p> <ul style="list-style-type: none"> Single-dimension arrays take less memory and execute faster than two-dimension or three-dimension arrays. Direct references to array elements execute faster than indexed references. An array can be as large as 2 MB. If you create an array of structures, the memory for each element is allocated based on the structure definition. 	
Type of Array	Benefit	Considerations
Single (1) dimension	<ul style="list-style-type: none"> Better support by native file instructions Fully supported in user-defined structures and arrays Smallest impact (execution time and memory) for indexed references Can create arrays when programming online 	<ul style="list-style-type: none"> Multiple arrays cannot be indirectly referenced like in PLC or SLC™ processors (such as, N(N7:0):5) BOOL arrays are not directly supported by file instructions Can be changed only when programming offline
Double (2) dimension and Triple (3) dimension	<ul style="list-style-type: none"> Can provide a more accurate data representation for a physical system Can emulate PLC file/word indirection with a two-dimension array Can create arrays when programming online 	<ul style="list-style-type: none"> Larger impact (execution time and memory) for indexed references File manipulation requires extra code and file instructions Can only be changed when programming offline
Nest arrays.	The file instructions offer limited support for arrays. To work with array data, create a user-defined structure with one array as a member of the structure. Then create an array tag by using the user-defined structure as its data type.	
Select the data type of the array based on the data and the instructions that manipulate that data.	While SINT and INT arrays can compact more values into a given memory area, they require additional memory and execution time for each instruction that references the array.	
Limit arrays to 2 MB of data.	The maximum array size is 2 MB. The software displays a warning if you try to create an array that is too large. The software also displays a warning if an array is 1.5...2 MB, even though these sizes are valid.	
Edit arrays online and offline.	You can create arrays when online or offline. However, you can modify only the size or data type of an existing array when offline.	

Indirect Addresses of Arrays

If you want an instruction to access different elements in an array, use a tag in the subscript of the array (an indirect address). By changing the value of the tag, you change the element of the array that your logic references.

When *index* equals 1, *array[index]* points here.



When *index* equals 2, *array[index]* points here.

When you directly reference an element in an array (such as `MyArray[20]`), uses less memory and executes faster than an indirect reference (`MyArray[MyIndex]`). You can also indirectly address bits in a tag (`MyDint.[Index]`).

If you use indirect addresses, use DINT tags because other data types require conversion and execute slower. For each indexed access to data, the controller recalculates the array index. If you access a specific array element multiple times, copy the data out of the array into a fixed tag and use that tag in subsequent logic.

You can also use an expression to specify the index value. For example: `MyArray[10 + MyIndex]`.

- An expression uses operators to calculate a value.
- The controller computes the result of the expression and uses it as the index.
- These are valid operators.

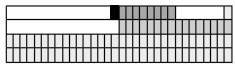
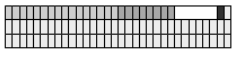
Operator	Description	Optimal
+	Add	DINT, REAL
-	Subtract/negate	DINT, REAL
*	Multiply	DINT, REAL
/	Divide	DINT, REAL
**	Exponent (x to y)	DINT, REAL
ABS	Absolute value	DINT, REAL
ACS	Arc cosine	REAL
AND	Bitwise AND	DINT
ASN	Arc sine	REAL
ATN	Arc tangent	REAL
COS	Cosine	REAL
DEG	Radians to degrees	DINT, REAL
FRD	BCD to integer	DINT

Operator	Description	Optimal
LN	Natural log	REAL
LOG	Log base 10	REAL
MOD	Modulo divide	DINT, REAL
NOT	Bitwise complement	DINT
OR	Bitwise OR	DINT
RAD	Degrees to radians	DINT, REAL
SIN	Sine	REAL
SQR	Square root	DINT, REAL
TAN	Tangent	REAL
TOD	Integer to BCD	DINT
TRN	Truncate	DINT, REAL
XOR	Bitwise exclusive OR	DINT

Guidelines for Array Indexes


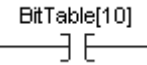
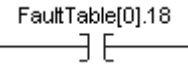

Guideline	Description								
Use the SIZE instruction to determine the number of elements in an array.	<p>By determining the number of elements in an array at runtime, you can write reusable code that adjusts itself to meet each instance where it is used.</p> <div style="border: 1px solid black; padding: 10px; margin: 10px auto; width: fit-content;"> <p style="text-align: center;">SIZE</p> <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="border-right: 1px solid black; padding: 2px 10px;">Size in Elements</td><td style="padding: 2px 10px;">?</td></tr> <tr> <td style="border-right: 1px solid black; padding: 2px 10px;">Source</td><td style="padding: 2px 10px;">??</td></tr> <tr> <td style="border-right: 1px solid black; padding: 2px 10px;">Dim. To Vary</td><td style="padding: 2px 10px;">?</td></tr> <tr> <td style="border-right: 1px solid black; padding: 2px 10px;">Size</td><td style="padding: 2px 10px;">??</td></tr> </table> </div> <p>The SIZE instruction returns the number of elements. Arrays are zero-based, so subtract 1 from the result to determine the last element position.</p>	Size in Elements	?	Source	??	Dim. To Vary	?	Size	??
Size in Elements	?								
Source	??								
Dim. To Vary	?								
Size	??								
Use immediate values to reference array elements.	Immediate value references to array elements are quicker to process and execute faster than indexed references.								
Use DINT tags for array indexes.	DINT tags execute the fastest. SINT, INT, and REAL tags require conversion code that can add additional scan time to an operation.								
Avoid using array elements as indexes.	The controller does not directly support the use of an array element as the index to look up a value in another array. To work around this, you can create an alias to the element and then use this as the index. Or copy the element to a base tag and use that base tag as the index.								

Guidelines for User-defined Data Types (UDT)

Guideline	Description
Group members of the same data type within a structure.	<p>You can create members of most data types, except for AXIS, COORDINATE_SYSTEM, MOTION_GROUP, and MESSAGE data types.</p> <p>Place members that use the same data type in sequence.</p> <div style="display: flex; justify-content: space-around; align-items: flex-start;"> <div style="text-align: center;"> <p>16 bytes</p>  <p>DATATYPE Sample1 BOOL Bit1; SINT Tiny_Value BOOL Bit2; INT Small_Value DINT Big_Value REAL Float_Value END_TYPE</p> </div> <div style="text-align: center;"> <p>12 bytes</p>  <p>DATATYPE Sample1 BOOL Bit1; BOOL Bit2; SINT Tiny_Value INT Small_Value DINT Big_Value REAL Float_Value END_TYPE</p> </div> </div> <p>The controller aligns every data type along an 8-bit boundary for SINTs, a 16-bit boundary for INTs, or a 32-bit boundary for DINTs and REALs. BOOLs also align on 8-bit boundaries, but if they are placed next to each other in a user-defined structure, they are mapped so that they share the same byte.</p>
Arrays within a UDT can only be 1-dimension.	If you include an array as a member, limit the array to one dimension. Multidimension arrays are not permitted in a user-defined structure.
I/O data that is used in a UDT must be copied into the members.	<p>If you include members that represent I/O devices, you must use logic to copy the data into the members of the structure from the corresponding I/O tags.</p> <p>Make sure that the data type of the structure member matches the I/O data type to avoid data type conversion.</p>
Limit user-defined data types to 500 members.	The controllers limit user-defined structures to 500 members. If you need more, consider nesting structures within the main structure.
Limit user-defined data types to 2 MB of data.	The maximum UDT size is 2 MB. The software displays a warning if you try to create a UDT that is too large. The software also displays a warning if the UDT is 1.5...2 MB, even though these sizes are valid.
Limit the size of user-defined structures if they are to be produced and consumed tags.	<p>Produced and consumed tags are limited to 500 bytes over the backplane and 480 bytes if over a network.</p> <p>Linux-based software can optimize user-defined structures that are less than 480 bytes. UDT larger than the noted produced and consumed tag limits must use a MESSAGE instruction (MSG) if they are to be communicated.</p>
Use the appropriate instruction to load data into a structure.	<p>Load input values into the user-defined structure at the beginning of the program and copy output values from the user-defined structure at the end of the program.</p> <ul style="list-style-type: none"> • Single bit - Examine On (XIC) and Output Energize (OTE) instructions • Contiguous bits - Bit Field Distribute (BTD) instruction • Single value - MOV instruction • Multiple contiguous values -COP/CPS instruction
Use structure descriptions to automatically create tag descriptions.	Enable the Use Pass-through Description workstation option (Tools > Options > Display) to display the descriptions you add to the members of structures for each tag that uses that structure data type.
Online and offline editing.	You can create user-defined structures when online or offline. However, you can modify only an existing structure when offline.

Select a Data Type for Bit Tags

Bits in a controller can exist as: BOOL tags, bits in a BOOL array, bits in elements of a SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT array, members of a user-defined structure, or as bits in a SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT member of a user-defined structure.

Tag Type	Description	
BOOL tag MyBit:BOOL 	Each tag accesses a specific bit. Each tag uses 4 bytes.	
	Benefits <ul style="list-style-type: none"> Each bit has a specific tag 	Considerations <ul style="list-style-type: none"> Requires extra bandwidth to communication Uses more memory Cannot use FBC/DDT bit file instructions
BOOL array BitTable:BOOL[32] 	A BOOL array combines multiple bits into adjacent words (32-bit words).	
	Benefits <ul style="list-style-type: none"> Consolidates multiple bits into one word Better use of memory Can address all bits in an array by using indirect addressing 	Considerations <ul style="list-style-type: none"> BOOL data type only supported by bit instructions Cannot use file instructions, copy instructions, or DDT/FBC instructions
DINT array FaultTable:DINT[3] 	A DINT combines multiple bits into adjacent words.	
	Benefits <ul style="list-style-type: none"> Consolidates multiple bits into one word File instructions, copy instructions, and DDT/FBC instructions support DINT arrays Lets you access the bits by element (word) and bit number 	Considerations <ul style="list-style-type: none"> Requires extra planning to indirectly address bits Difficult to address bits in the array by using indirect addressing
User-defined structure BitStructure Bit1:BOOL Bit2:BOOL 	A user-defined structure combines multiple bits into adjacent, individually named words.	
	Benefits <ul style="list-style-type: none"> Object based Consolidates multiple bits into one word 	Considerations <ul style="list-style-type: none"> Third-party MMI/EOI products do not directly support structures. Cannot use FBC/DDT bit file instructions

Serial Bit Addresses

The BOOL B data table in the PLC-5® and SLC™ 500 processors supports two address modes that can address the same bit.

Address Mode	Description
Serial bit In PLC-5 or SLC software, this addressing mode is represented as /Bit	Serial bit addressing references all bits as a contiguous list (array) of bits. For example, if you want to reference the third bit in the second word of a B file, specify B3/18. This method is similar to a BOOL array in a Logix 5000™ controller where you specify FaultBit[18].
Word bit In PLC-5 or SLC software, this addressing mode is represented as Word/Bit	Word bit addressing identifies a bit within a specific word. For example, B3:1/2 is the same as B3/18 from the serial bit example. This method is similar to accessing the bits of a SINT, INT, DINT array in a Logix 5000 controller where you specify FaultTable[1].2.

The Logix 5000 controller supports both of these addressing modes, but you cannot use both to reference bits in the same array due to conformance with the IEC 61131-3 standard. Choose the method that best meets your application needs. You can copy data between arrays by using both methods.

You can also use an expression to indirectly reference a bit in a DINT array by using a serialized bit number. For example:

Tag

MyBits : DINT[10]

BitRef : DINT

EndTag

MOV(34, BitRef)

XIC(MyBits[BitRef / 32].[BitRef AND 31])

where:

This expression	Calculates the
[BitRef / 32]	Element in the DINT array
If the tag MyBits is an INT or SINT, the divisor is 16 or 8, respectively.	
[BitRef AND 31]	Bit within the element
If the tag MyBits is an INT or SINT, the mask value is 15 or 7, respectively.	

The Diagnostic Detect (DDT) and File Bit Compare (FBC) instructions provide a bit number as a result of their operation. These instructions are limited to DINT arrays so you can use them to locate the bit number that is returned from the example above.

Guidelines for String Data Types

String data types are structures that hold ASCII characters. The first member of the structure defines the length of the string; the second member is an array that holds the actual ASCII characters.

Name:emailString

Description:string type used for email source/destination data

Maximum Characters:512

Members:

	Name	Data Type	Style	Description
	LEN	DINT	Decimal	
	DATA	SINT[512]	ASCII	

Data Type Size: 516

Guideline	Description
You can create a string data type that is longer or shorter than the default string data type.	The default string data type can contain as many as 82 characters. You can create custom-length string data types that range from 1 to 65535 characters.
Only some instructions support string data types.	These comparison instructions support string tags: EQU, NEQ, GRT, GEG, LES, LEQ, CMP. These serial port instructions support string tags: ARD, ARL, AWA, AWT. These string-handling instructions support string tags: STOD, DTOS, STOR, RTOS, CONCAT, MID, FIND, DELETE, INSERT, UPPER, LOWER, SIZE. These file instructions support string arrays: FAL, FFL, FFU, LFL, LFU, COP, CPS, FSC.
Use the SIZE instruction to determine the number of characters in a string.	By determining the number of characters in a string at runtime, you can write reusable code that adjusts itself to meet each instance where it is used.
Use the DTOS, RTOS, and CONCAT instructions to embed tag values within a string.	The SLC 500 processor supports the ability to embed a data-table reference address within a string (inline indirection). The SLC 500 AWA and AWT instructions can then look up the data value and place an ASCII representation into the outgoing string. The Logix 5000 controller does not directly support this ability. Use the DTOS or RTOS instructions to convert a value to a string and the CONCAT instruction to merge characters with another string.
Set the LEN field to indicate the number characters that are present.	The LEN field in the string structure indicates how many characters are in the string. The programming software and the controller instructions that manipulate strings use the LEN value to determine how many positions in the string DATA array contain valid characters. Both the programming software and the instructions stop processing the DATA array once they reach the LEN value.

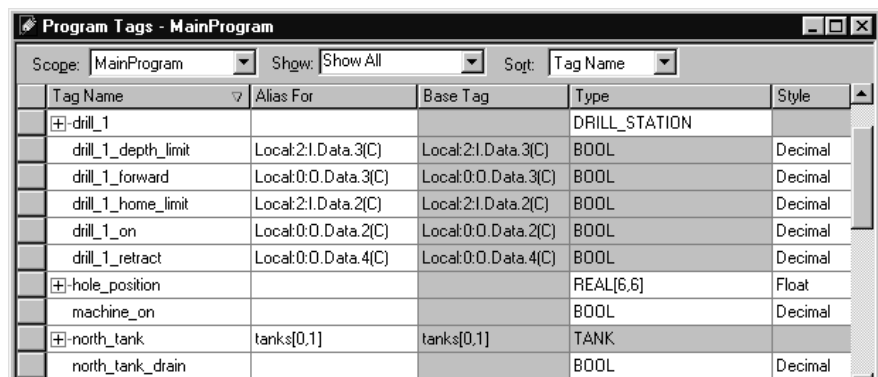
PLC-5/SLC 500 Access of Strings

The ASCII A data table in the PLC-5 and SLC 500 processors uses a string format that is similar to the Logix string data type. The main difference is that the LEN field (length) in a PLC-5/SLC 500 processor is a 16-bit, INT value. The LEN field in a Logix 5000 controller is a 32-bit, DINT field. This difference can impact converted logic and data communication. The Logix 5000 controller converts the LEN field to the appropriate value and size when a PLC-5/SLC 500 message format is used to read or write a string.

Configure Tags

A tag is a text-based name for an area of controller memory where data is stored. Tags are the basic mechanism to allocate memory, reference data from logic, and monitor data.

If you want the tag to	Then choose this type
Store a value for use by logic within the project	Base
Use another name for an existing tag's data (can help simplify long, pre-determined tag names, such as for I/O data or user-defined structures)	Alias
Send (broadcast) data to another controller	Produced
Receive data from another controller	Consumed



Tag Name	Alias For	Base Tag	Type	Style
[-]drill_1			DRILL_STATION	
drill_1_depth_limit	Local:2:I.Data.3(C)	Local:2:I.Data.3(C)	BOOL	Decimal
drill_1_forward	Local:0:0.Data.3(C)	Local:0:0.Data.3(C)	BOOL	Decimal
drill_1_home_limit	Local:2:I.Data.2(C)	Local:2:I.Data.2(C)	BOOL	Decimal
drill_1_on	Local:0:0.Data.2(C)	Local:0:0.Data.2(C)	BOOL	Decimal
drill_1_retract	Local:0:0.Data.4(C)	Local:0:0.Data.4(C)	BOOL	Decimal
[-]hole_position			REAL[6,6]	Float
machine_on			BOOL	Decimal
[-]north_tank	tanks[0,1]	tanks[0,1]	TANK	
north_tank_drain			BOOL	Decimal

For more information on I/O tags, see [Communicate with I/O on page 89](#).

Guidelines for Base Tags

Use the following guidelines for base tags.

Guideline	Description
Create standalone atomic tags.	<p>The controller supports pre-defined, standalone tags.</p> <ul style="list-style-type: none"> Atomic tags are listed directly in the Tag Editor and Data Monitor and can easily be found by browsing the alphabetical list. Atomic tags can be created online, but the data type can be only modified offline. <p>Using only atomic tags can impact HMI communication performance as more information must be passed and acted on.</p>
Create user-defined structures	<p>User-defined structures (data types) let you organize your data to match your machine or process.</p> <ul style="list-style-type: none"> One tag contains all data that is related to a specific aspect of your system. This keeps related data together and easy to locate, regardless of its data type. Each piece of data (member) gets a descriptive name. You can use the structure to create multiple tags with the same data layout. User-defined structure can only be modified offline. <p>Linux-based software optimizes user-defined structures more than standalone tags.</p>
Use arrays like files to create a group of similar tags.	<p>An array creates multiple instances of a data type under a common tag name.</p> <ul style="list-style-type: none"> Arrays let you organize a block of tags that use the same data type and perform a similar function. You organize the data in one, two, or three dimensions to match what the data represents. Arrays can be only modified offline. Linux-based software optimizes array data types more than standalone tags. <p>Minimize the use of BOOL arrays. Many array instructions do not operate on BOOL arrays, making it more difficult to initialize and clear an array of BOOL data.</p>
Take advantage of program-scoped tags.	<p>If you want multiple tags with the same name, define each tag at the program scope (program tags) for a different program. This lets you reuse both logic and tag names in multiple programs.</p> <p>Avoid using the same name for both a controller tag and a program tag. Within a program, you cannot reference a controller tag if a tag of the same name exists as a program tag for that program.</p>

Guideline	Description
Use mixed case and the underscore characters.	Although tags are not case-sensitive (upper case A is the same as lower case a), mixed case is easier to read. For example, Tank_1 can be easier to read than tank1.
Consider alphabetical order.	The programming software displays tags of the same scope in alphabetical order. To make it easier to monitor related tags, use similar starting characters for tags that you want to keep together. For example, consider using Tank_North and Tank_South rather than North_Tank and South_Tank.
Use leading zeroes (0) when numbers are part of tag names	The programming software uses a simple sort to alphabetize tag names in the Tag Editor and Data Monitor. This means if you have Tag1, Tag2, Tag11, and Tag12, the software displays them in order as Tag1, Tag11, Tag12, and then Tag2. If you want to keep them in numerical order, name them Tag01, Tag02, Tag11, and Tag12.

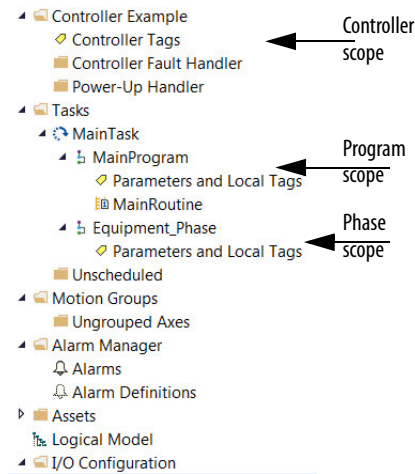
Create Alias Tags

An alias tag lets you create one tag that represents another tag.

- Both tags share the same value as defined by the base tag.
- When the value of a base tag changes, all references (aliases) to the base tag reflect the change.

Guideline	Description
Upload behavior for alias tags.	For 5370 and 5570 controllers, there are situations when uploading a project file that instruction operands that use alias references can change. To avoid these situations with ladder instruction operands, do not use: <ul style="list-style-type: none"> • Nested aliases (also known as an alias chain) • Multiple aliases to the same tag On an upload, the software uses a technique referred to as “best fit” to reconstruct instruction operands and operands that use an alias can change. This does not apply to 5380, 5480 and 5580 controllers and on an upload, the aliases for these controllers are faithfully reproduced.
Alias tags do not affect controller execution.	During download, the program is compiled into machine executable code and physical memory addresses. While the existence of an alias requires controller memory to store the name, the program performs the same operation for a reference with an alias or its associated base tag.
Access alias tags from Linux-based software.	Because an alias tag appears as a standalone tag to Linux-based software, an alias tag that references a compound array or structure can require additional communication time. When you reference tags from Linux-based software or other HMI, it can be fastest to reference base tags directly.

Guidelines for Data Scope



Data scope defines where you can access tags. Controller-scoped tags and parameters are accessible by all programs. Local tags are accessible only by the code within a specific program. Equipment Phases, like Programs, have parameters and local tags.

If you want to	Then assign this scope
Produce or consume data	Controller scope (controller tags)
Use a tag in multiple programs in the same project	Controller scope (controller tags)
Use a tag in a message (MSG) instruction	Parameters
Use motion tags	Parameters
Reuse the same tag name multiple times for different parts or processes within a controller	Local Tags
Have multiple programmers work on logic and you want to merge logic into one project	Local Tags

Isolate portions of a machine or different stations into separate programs or equipment phases and use program-scoped or phase-scoped tags. This lets you do the following:

- Provide isolation between programs and equipment phases
- Help prevent tag name collisions
- Improve the ability to reuse code

See publication [1756-PM021, Logix 5000 Controllers Program Parameters Programming Manual](#), for more information on Parameters.

Guidelines for Tag Names

Use the following guidelines when you name tags.

Guideline	Description
Create descriptive names but keep them short.	<p>Tag names can be from 1...40 characters long.</p> <ul style="list-style-type: none"> • Each character of the tag name uses 1 byte of controller memory, rounded to a 4-byte boundary. • For example, a tag name with 1...4 characters uses 4 bytes. A tag name with 5 characters uses 8 bytes. • Tag names are stored in the controller. • Use structures to reduce the number and size of tags needed. <p>Program upload preserves tag names.</p>
Create a naming convention.	Develop a tag-naming convention on electrical drawings or machine design. For example, Conv1_Full_PE101 combines the sensor function with the photoeye number.
Use correct characters in tag names.	<p>Tag names follow the IEC 61131-3 standard. You can use:</p> <ul style="list-style-type: none"> • Letters A through Z. • Numbers 0...9. • Underscore character (_). <p>Tags must start with a letter to avoid confusion with logical expressions. The remaining characters can be any of the supported characters.</p>
Pad names to improve sort order.	<p>The programming software displays tags in alphabetical order. If you use numbers in your tag names, pad the number with leading zeros so the names sort in the proper order.</p> <p>For example, tag names: TS1, TS2, TS3, TS10, TS15, TS20, TS30 display as: TS1, TS10, TS15, TS2, TS20, TS3, and TS30.</p> <p>Pad the numbers with zero so they display as: TS01, TS02, TS03, TS10, TS15, TS20, TS30.</p>

Guidelines for Extended Tag Properties

Use the following guidelines for extended tag properties.

Guideline	Description
Use extended tag properties to define additional information, such as limits, engineering units, or state identifiers, for various components within your controller project.	You can define extended tag properties for these components: <ul style="list-style-type: none"> • Tag • Parameter • User-defined data type • Add-On Instruction
Some extended tag properties support pass-through for data structures and arrays.	Pass-through behavior is available for descriptions, state identifiers, and engineering units and is configurable in data structures and arrays. Pass-through behavior is not available for limits.
You can read extended properties via logic, but you cannot write to extended properties values in logic.	<ul style="list-style-type: none"> • Extended properties must be used as an input operand. • Alias tags with extended properties cannot be accessed in logic. • Limits can be configured for input and output parameters in Add-On Instructions. However, limit extended properties must not be defined on an InOut parameter of an Add-On Instruction. • Limits cannot be accessed inside Add-On Instruction logic. • If you read an extended property value in logic, it consumes memory equivalent to an equivalent program-scoped tag of that data type. If you do not use them in logic, extended tag properties use no user memory, only extended memory.
If an array tag uses indirect addressing to access limit extended properties in logic, the following conditions apply.	<ul style="list-style-type: none"> • If the array tag has limit extended properties that are configured, the extended properties are applied to any array element that does not explicitly have that particular extended property configured. For example, if the array tag MyArray has Max configured to 100, then any element of the array that does not have Max configured inherits the value of 100 when used in logic. However, it is not visible to you that the value inherited from MyArray is configured in the tag properties. • At least one array element must have specific limit extended property configured for indirectly referenced array logic to verify. For example, if MyArray[x].@Max is being used in logic, at least one array element of MyArray[] must have Max extended property configured if Max is not configured by MyArray. • Under the following circumstances a data type default value is used: <ul style="list-style-type: none"> – Array is accessed programmatically with an indirect reference. – Array tag does not have the extended property configured. – A member of an array does not have the extended property configured.

Tag Descriptions

The programming software searches a tag's origin to locate the first available description. This reduces the number of descriptions you need to enter. This also verifies that tag references display associated descriptions.

Guideline	Description												
Tag descriptions display in the programming software according to the tag's origin.	<table> <tr> <th>Type of Tag</th><th>Description Display in the programming software</th></tr> <tr> <td>Atomic</td><td>For a BOOL, SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT, REAL or LREAL tag, the description that is associated with the tag is the only description available for display.</td></tr> <tr> <td>Alias</td><td>First the alias tag description, then the base tag description.</td></tr> <tr> <td>User-defined structure and Add-On Instruction</td><td>All members use the description for tag, unless you define a specific description for a member. For example, MyTimer.DN uses the description for MyTimer if there is no description for MyTimer.DN.</td></tr> <tr> <td>Atomic array</td><td> <ul style="list-style-type: none"> • All references into an array use the description for the array, unless you define a description for an element of the array. • For example, MyTable[10] uses the description for MyTable if there is no description for MyTable[10]. • All indexed references into an array use the description for the array. • For example, MyTable[Index] uses the description for MyTable. </td></tr> <tr> <td>Structure array</td><td>All references to a member of a structure in an array default to the array definition, unless you define a description for the structure member of the array. For example, Table[0].Field1 uses the description for Table if there is no description for the specific field.</td></tr> </table>	Type of Tag	Description Display in the programming software	Atomic	For a BOOL, SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT, REAL or LREAL tag, the description that is associated with the tag is the only description available for display.	Alias	First the alias tag description, then the base tag description.	User-defined structure and Add-On Instruction	All members use the description for tag, unless you define a specific description for a member. For example, MyTimer.DN uses the description for MyTimer if there is no description for MyTimer.DN.	Atomic array	<ul style="list-style-type: none"> • All references into an array use the description for the array, unless you define a description for an element of the array. • For example, MyTable[10] uses the description for MyTable if there is no description for MyTable[10]. • All indexed references into an array use the description for the array. • For example, MyTable[Index] uses the description for MyTable. 	Structure array	All references to a member of a structure in an array default to the array definition, unless you define a description for the structure member of the array. For example, Table[0].Field1 uses the description for Table if there is no description for the specific field.
Type of Tag	Description Display in the programming software												
Atomic	For a BOOL, SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT, REAL or LREAL tag, the description that is associated with the tag is the only description available for display.												
Alias	First the alias tag description, then the base tag description.												
User-defined structure and Add-On Instruction	All members use the description for tag, unless you define a specific description for a member. For example, MyTimer.DN uses the description for MyTimer if there is no description for MyTimer.DN.												
Atomic array	<ul style="list-style-type: none"> • All references into an array use the description for the array, unless you define a description for an element of the array. • For example, MyTable[10] uses the description for MyTable if there is no description for MyTable[10]. • All indexed references into an array use the description for the array. • For example, MyTable[Index] uses the description for MyTable. 												
Structure array	All references to a member of a structure in an array default to the array definition, unless you define a description for the structure member of the array. For example, Table[0].Field1 uses the description for Table if there is no description for the specific field.												

For more information, see the Create Tag Descriptions Automatically with User-Defined Data Types White Paper, publication [LOGIX-WP004](#).

Protect Data Access Control at Tag Level

New tag attributes define access to tag data at runtime.

Tag Attribute	Description
External access	Defines how an external application, such as an HMI, historian, or OPC data server, can access a tag. For arrays, this feature applies to the top level only; for user-defined structure, this feature applies to individual members. Possible values are: <ul style="list-style-type: none">• Read/Write: External applications can both read and modify the tag's value• Read Only: External applications can read the tag's value, but not modify it• None: External applications can neither read or write the tag's value
Constant	Defines whether a tag value remains constant. Tags with this attribute set cannot be changed programmatically.

You can Use RSLinx® Classic software, version 2.56 or later, RSLinx Enterprise software, versions 5.21 to 5.90, or FactoryTalk® Linx software version 6.00 or later, for best results with these tag attributes. Using earlier versions of RSLinx software can result in anomalous behavior from the data servers with the external access options of Read Only and None.

Notes:

Communicate with I/O

I/O values update at a period, requested packet interval (RPI), which you configure via Module Property dialog in the I/O configuration folder of the project. The values update asynchronously to the execution of logic.

The module sends input values to the controller at the specified RPI. Because this transfer is asynchronous to the execution of logic, an I/O value in the controller can change in the middle of a scan.

Buffer I/O Data

If you reference an I/O tag multiple times, and the application could be impacted if the value changes during a program scan, you must buffer the I/O value. You can buffer an I/O tag by using input parameters or copying into a buffer tag. In your code, reference the buffer tag rather than the I/O tag.

IMPORTANT	Use the synchronous copy (CPS) instruction to buffer I/O data. While the CPS instruction copies data, no I/O updates or other tasks can change the data. Tasks that attempt to interrupt a CPS instruction are delayed until the instruction is done. Overuse of the CPS instruction can impact controller performance by keeping all other tasks from executing.
------------------	---

Buffer I/O data to do the following:

- Help prevent an input or output value from changing during the execution of a program (I/O updates asynchronous to the execution of logic).
- Copy an input or output tag to a member of a structure or element of an array.
- Help prevent produced or consumed data from changing during the execution of a program.
- Make sure all produced and consumed data arrives or is sent as a group (not mixed from multiple transfers).
- Only use the CPS instruction if the I/O data that you want to buffer is greater than 32 bits (or 4 bytes) in size.

Overuse of the CPS instruction can greatly impact controller performance.

If you have a user-defined structure with members that represent I/O devices, you must use logic to copy the data into the members of the structure from the corresponding I/O tags.

Guidelines to Specify an RPI Rate for I/O Modules

Configure an RPI rate per module (ControlLogix®) or an RPI rate per controller (CompactLogix™). The RPI value is the rate at which the controller attempts to communicate with the module.

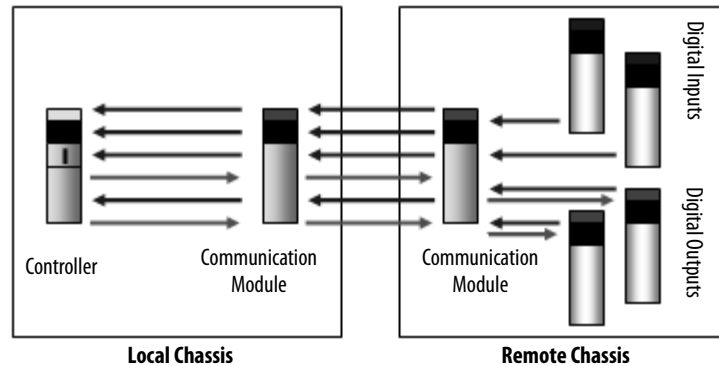
Guideline	Description
Specify an RPI at 50% of the rate you need new data.	If you set the RPI faster (specify a smaller number) than what your application needs, it can waste network resources, such as ControlNet® schedule bandwidth, network processing time, and CPU processing time. For example, if you need information every 80 ms, set the RPI at 40 ms. The data is asynchronous to the controller scan, so you sample data twice as often (but no faster), you ensure that you have the most current data.
Group devices with similar performance needs onto the same module.	By grouping devices with similar performance needs on the same module, you consolidate data transmission to one module rather than multiple modules. This conserves network bandwidth.
Set the ControlNet™ network update time (NUT) equal to or less than the fastest RPI.	When configuring a ControlNet network, set the network update time (NUT) equal to or less than the fastest RPI of the I/O modules and produced/consumed tags in the system. For example, if your fastest RPI is 10 ms, set the NUT to 5 ms for more flexibility in scheduling the network.
In an ControlNet system, use even multiples of the NUT for the RPI value.	Set the RPI to a binary multiple of the NUT. For example, if the NUT is 10 ms, select an RPI such as 10, 20, 40, 80, or 160 ms.
In a ControlNet system, isolate I/O communication.	If you use unscheduled ControlNet communication or want to be able to add ControlNet I/O at runtime (see page 98), dedicate one ControlNet network to I/O communication only. On the dedicated I/O network, make sure that there is the following: <ul style="list-style-type: none"> • No HMI traffic • No MSG traffic • No programming workstations • No peer-to-peer interlocking in multi-processor system architectures
In an EtherNet/IP™ system, module change of state is limited to 1/4 of the RPI.	If you configure change of state communication for a module in a remote chassis that is connected via an EtherNet/IP network, the module can send data only as fast as the module RPI. Initially, the module sends its data immediately. However, when an input changes, the module data is held at the adapter until 1/4 of the RPI is reached to avoid overloading the EtherNet/IP network with the module communication.
Data transmission depends on the controller.	The type of controller determines the data transmission rate. <ul style="list-style-type: none"> • ControlLogix controllers transmit data at the RPI you configure for the module. • CompactLogix controllers transmit data at powers of 2 ms (such as 2, 4, 8, 16, 64, or 128). For example, if you specify an RPI of 100 ms, the data actually transfers at 64 ms.

Communication Formats for I/O Modules

The communication format determines whether the controller connects to the I/O module via a direct or a rack-optimized connection. The communication format also determines the type and quantity of information that the module provides or uses.

Direct Connection

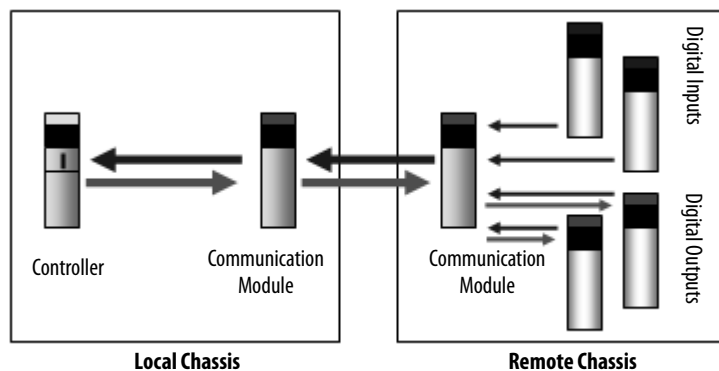
Each module passes its data to/from the controller individually. Communication modules bridge data across networks.



Benefits	Considerations
<ul style="list-style-type: none"> Each module can determine its own rate (RPI) More data can be sent per module, such as diagnostic and analog data Supports event task communication 	<ul style="list-style-type: none"> Requires additional connections and network resources This is the only method that is supported in the local chassis I/O data is presented as individual tags

Rack-optimized Connection

The communication module in a remote chassis consolidates data from multiple modules into one packet and transmits that packet as one connection to the controller.



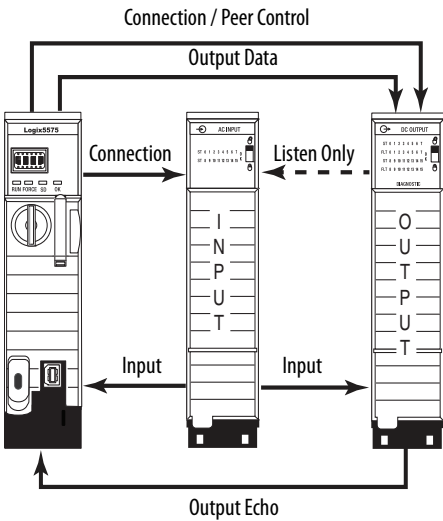
Benefits	Considerations
<ul style="list-style-type: none"> One connection can service a full chassis of digital modules Reduces network resources and loading 	<ul style="list-style-type: none"> All Modules are sent at the same rate Unused slots are still communicated Still need a direct connection for analog and diagnostic data Limited to remote chassis I/O data is presented as arrays with alias tags for each module

The rack-optimized format limits data to one 32-bit input word per module in a chassis. If you place a diagnostic module in a chassis, the rack-optimized format eliminates the value that the diagnostic module offers. In this case, it's better to use a direct connection so that the diagnostic information from the module is passed to the controller.

Peer Control Output modules let peer ownership of input modules to consume input data to directly control outputs without requiring controller processing.

The 1756-IB16IF and 1756-IB16IFC modules can be listened to presuming the output module knows the input data layout and connection information. The configuration from the controller defines how the peer input data is mapped to the output modules. The controller can use the other digital points on the module that are not peer-owned as conventional outputs.

The controller can also use the output data it normally sends to the module with consumed inputs, letting ‘gate-type’ features enabled by controller logic selectively letting application of the consumed peer input data.



Benefits	Considerations
<ul style="list-style-type: none">• Faster response time because the controller scan time is removed from the equation. Data is sent directly to the output module from the input module.• Increases controller performance by reducing the need for event tasks to close loops quickly.• Each input module has an AND and OR bit mask that defines the logic that is applied to each input module.	<ul style="list-style-type: none">• You must program the controller for proper relationship with the output modules.• The peer output module must be in the same chassis as the input module to maximize response time.

Electronic Keying

Electronic Keying reduces the possibility that you use the wrong device in a control system. It compares the device that is defined in your project to the installed device. If keying fails, a fault occurs. These attributes are compared.

Attribute	Description
Vendor	The device manufacturer.
Device Type	The general type of the product, for example, digital I/O module.
Product Code	The specific type of the product. The Product Code maps to a catalog number.
Major Revision	A number that represents the functional capabilities of a device.
Minor Revision	A number that represents behavior changes in the device.

The following Electronic Keying options are available.

Keying Option	Description
Compatible Module	Lets the installed device accept the key of the device that is defined in the project when the installed device can emulate the defined device. With Compatible Module, you can typically replace a device with another device that has the following characteristics: <ul style="list-style-type: none"> • Same catalog number • Same or higher Major Revision • Minor Revision as follows: <ul style="list-style-type: none"> – If the Major Revision is the same, the Minor Revision must be the same or higher. – If the Major Revision is higher, the Minor Revision can be any number. This is the default selection in the Logix Designer application.
Disable Keying	Indicates that the keying attributes are not considered when attempting to communicate with a device. With Disable Keying, communication can occur with a device other than the type specified in the project. ATTENTION: Be cautious when using Disable Keying; if used incorrectly, this option can lead to personal injury or death, property damage, or economic loss. We strongly recommend that you do not use Disable Keying. If you use Disable Keying, you must take full responsibility for understanding whether the device being used can fulfill the functional requirements of the application.
Exact Match	Indicates that all keying attributes must match to establish communication. If any attribute does not match precisely, communication with the device does not occur.

Carefully consider the implications of each keying option when selecting one.

IMPORTANT When you change Electronic Keying parameters online, it interrupts connections to the device and any devices that are connected through the device. Connections from other controllers can also be broken.

If an I/O connection to a device is interrupted, the result can be a loss of data.

More Information

For more detailed information on Electronic Keying, see Electronic Keying in Logix 5000™ Control Systems Application Technique, publication [LOGIX-AT001](#).

Guidelines to Manage I/O Connections

Use the following guidelines to administer your I/O modules.

IMPORTANT Compact 5000™ I/O does not support rack-optimization.

1. The type of I/O module can determine the type of connection.
 - Analog modules always use direct connections, except for 1771 analog modules that use messaging and 1734 analog modules that use Enhanced Rack Optimization.
 - Digital modules can use direct or rack-optimized connections. Communication formats that include optimization in the title are rack-optimized connections; all other connection options are direct connections.
2. Select the communication format for a remote adapter based on the remote I/O modules.

Select	If
None	The remote chassis contains only analog modules, diagnostic digital modules, fused output modules, or communication modules. On a ControlNet network, use None to add a chassis to the network while the controller is running.
Rack-optimized	The remote chassis only contains standard, digital input, and output modules (no diagnostic modules or fused output modules). For a ControlNet network at runtime (controller is online), you can add new digital modules to an existing rack-optimized connection, but new rack-optimized connections can only be added when offline. An EtherNet/IP network supports new rack-optimized connections both offline and at runtime (online). For more information, see page 98.
Listen Only Rack-optimized	You want to receive I/O module and chassis slot information from a rack-optimized remote chassis that is owned by another controller. The runtime capability for listen only rack-optimized connections is the same as for rack-optimized connections.

3. Use rack-optimized connections to conserve connections.
If you are trying to limit the number of controller and network connections, rack-optimized connections can help.
4. In some cases, all direct connections work best.
For a remote adapter that is configured for rack-optimized connections, there is always data that is sent for each slot in the chassis, even if a slot is empty or contains a direct connection module. There are 12 bytes of data that is transferred for rack-optimized overhead between the controller and the remote adapter. In addition, the remote adapter sends 8 bytes per slot to the controller; the controller sends 4 bytes per slot to the remote adapter.

For a few digital modules in a large chassis, it can be better to use direct connections because transferring the full chassis information can require more system bandwidth than direct connections to a few modules.

Example	Description
Remote 17-slot chassis Slot 0: 1756-CNBR/D Slots 1...15: analog modules Slot 16: standard digital module	<p>Option 1: Select Rack Optimization as the communication format for the remote adapter. This example uses 16 controller connections (15 for analog modules and 1 for the rack-optimized connection). This example also transfers:</p> <ul style="list-style-type: none"> • 12 bytes for rack-optimized overhead. • 12 bytes for the digital module. • 12 bytes for each of the 15 analog modules, for a total of 180 bytes. <p>Option 2: Select None as the communication format for the remote adapter. This example also uses 16 controller connections (1 direct connection to each I/O module). There is no rack-optimized overhead data to transfer.</p> <p>Recommendation: Option 2 is recommended because it avoids unnecessary network traffic, and thus improves network performance.</p>
Remote 17-slot chassis Slot 0: 1756-CNBR/D Slots 1...8: analog modules Slots 9...16: digital modules	<p>Option 1: Select Rack Optimization as the communication format for the remote adapter. This example uses nine controller connections (eight for analog modules and one for the rack-optimized connection). This example also transfers:</p> <ul style="list-style-type: none"> • 12 bytes for rack-optimized overhead. • 12 bytes for each of the 8 digital modules, for a total of bytes 96 bytes. • 12 bytes for each of the 8 analog modules, for a total of 96 bytes. <p>Option 2: Select Rack Optimization for the communication format of the remote adapter. This example uses 16 controller connections (1 direct connection to each I/O module). There is no rack-optimized overhead data to transfer.</p> <p>Recommendation: The best option for this example depends on the type of digital I/O modules in the system and other controller connections. If the total system has many analog modules, diagnostic modules, fused output modules, or produced/consumed tags, select Option 1 to conserve controller connections. If there are plenty of controller connections available, select Option 2 to reduce unnecessary network traffic.</p>

Create Tags for I/O Data

Each I/O tag is automatically created when you configure the I/O module through the programming software. Each tag name follows this format:

Location:SlotNumber:Type.MemberName.SubMemberName.Bit

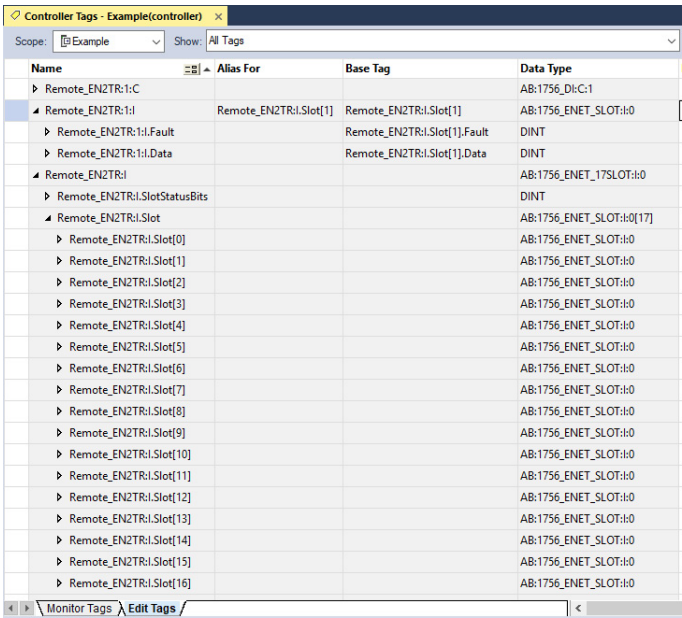
This address variable	Is
Location	Identifies network location LOCAL = local chassis or DIN rail ADAPTER_NAME = identifies remote adapter or bridge
SlotNumber	Slot number of I/O module in its chassis
Type	Type of data: I = input C = configuration O = output S = status
MemberName	Specific data from the I/O module, such as Data and Fault; depends on the module
SubMemberName	Specific data that is related to a MemberName
Bit (optional)	Specific point on the I/O module; depends on the size of the I/O module (0...31 for a 32-point module)

If you configure a rack-optimized connection, the software creates a rack-object tag for the remote communication module. You can reference the rack-optimized I/O module individually, or by its element within the rack-object tag.

For example, a remote EtherNet communication module (Remote_ENT2R) has an I/O module in slot 1.

The individual tag that is created for the I/O module in remote slot 1.

The entry in the rack-object tag for the remote communication module that identifies the I/O module in remote slot 1.



Controller Ownership

When you choose a communication format, you have to choose whether to establish an owner or listen-only relationship with the module.

Mode	Description
Owner	The owner controller writes configuration data and can establish a connection to the module.
Listen-only	A controller that uses a listen-only connection only monitors the module. It does not write configuration data and can only maintain a connection to the I/O module when the owner controller is actively controlling the I/O module.

There is a noted difference in the ownership of input modules versus the ownership of output modules.

Controlling	This Ownership	Description
Input modules	Owner	An input module is configured by a controller that establishes a connection as an owner. This configuring controller is the first controller to establish an owner connection. Once an input module has been configured (and owned by a controller), other controllers can establish owner connections to that module. This lets additional owners to continue to receive multicast data if the original owner controller breaks its connection to the module. All other additional owners must have the identical configuration data and identical communication format that the original owner controller has, otherwise the connection attempt is rejected.
	Listen-only	Once an input module has been configured (and owned by a controller), other controllers can establish a listen-only connection to that module. These controllers can receive multicast data while another controller owns the module. If all owner controllers break their connections to the input module, all controllers with listen-only connections no longer receive multicast data.
Output modules	Owner	An output module is configured by a controller that establishes a connection as an owner. Only one owner connection can be connected to an output module. If another controller attempts to establish an owner connection, the connection attempt is rejected.
	Listen-only	Once an output module has been configured (and owned by one controller), other controllers can establish listen-only connections to that module. These controllers can receive multicast data while another controller owns the module. If the owner controller breaks its connection to the output module, all controllers with listen-only connections no longer receive multicast data.

Runtime/Online Addition of Modules

You can add modules when the controller is in Run mode.

Network	Considerations																								
ControlNet network	<p>You can use:</p> <ul style="list-style-type: none">1756-CN2, 1756-CN2R, 1756-CN2RXT any series modules.1756-CNB, 1756-CNBR series D or later communication modules. <p>Digital I/O modules can be added as rack-optimized connections if the parent module is already configured with rack-optimized connections. While you can add a new digital I/O module to an existing rack-optimized connection, you cannot add rack-optimized connections while online. Digital I/O modules can also be added as direct connections.</p> <p>Analog I/O modules can be added only as direct connections.</p> <p>Disable the Change of State (COS) feature on digital input modules because it can cause inputs to be sent more quickly than the RPI.</p> <p>If you plan to add large amounts of I/O to the ControlNet network, dedicate one ControlNet network for I/O. For the dedicated ControlNet network, verify that there is little or no:</p> <ul style="list-style-type: none">HMI traffic.MSG traffic.Programming workstations. <p>If the module has a Real Time Sample (RTS), disable it or set to a rate that is greater than the RPI.</p> <p>Considerations for 1756-CN2, 1756-CN2R, 1756-CN2RXT Modules</p> <p>You can add I/O modules until you reach these limits:</p> <ul style="list-style-type: none">80% of CPU utilization of the 1756-CN2, 1756-CN2R, or 1756-CN2RXT communication module.Less than 400,000 unscheduled bytes per second are displayed in RSNetWorx™ for ControlNet software after the network has been scheduled. <p>Considerations for 1756-CNB, 1756-CNBR Modules</p> <p>Requested Packet Intervals (RPIs) faster than 25 ms for unscheduled modules can overload the 1756-CNB or 1756-CNBR communication module. To avoid the overload, make these considerations:</p> <ul style="list-style-type: none">Use a NUT of 10 ms or more.Keep the SMAX and UMAX values as small as possible. <p>You can add I/O modules until you reach these limits:</p> <ul style="list-style-type: none">75% of CPU utilization of the 1756-CNB or 1756-CNBR communication module.Plan for a CPU-use increase of 1...4% of the 1756-CNB or 1756-CNBR module for each I/O module you add, depending on RPI.48 connections on the 1756-CNB or 1756-CNBR communication module.Less than 400,000 unscheduled bytes per second are displayed in RSNetWorx for ControlNet software after the network has been scheduled.																								
EtherNet/IP network	<p>The EtherNet/IP I/O modules that you add at runtime can be:</p> <ul style="list-style-type: none">Added to existing rack-optimized connectionsAdded to new rack-optimized connectionsAdded as direct connections (you can create rack-optimized connections when adding EtherNet/IP I/O modules at runtime) <p>You can add I/O modules until you reach the limits of the communication module:</p> <table><tr><th>1756-EN4TR, 1756-EN4TRXT Module</th><th>1756-EN2TR, 1756-EN3TR</th><th>1756-EN2T, 1756-EN2TP, 1756-EN2TXT, 1756-EN2F Module</th><th>1756-ENBT Module</th><th>5069-AENTR, 5094-AENTxx COMPACT 5000 I/O Ethernet Adapter</th><th>5069-AEN2TR COMPACT 5000 I/O Ethernet Adapter</th></tr><tr><td><ul style="list-style-type: none">50,000 pps without CIP Security25,000 pps with integrity15,000 pps with integrity and confidentiality</td><td>20,000 pps</td><td>10,000 pps</td><td>5000 pps</td><td>100,000 pps (total number of packets from both Ethernet Ports)</td><td>100,000 pps (total number of packets from both Ethernet Ports)</td></tr><tr><td>512 TCP connections</td><td>128 TCP connections</td><td>128 TCP connections</td><td>64 TCP connections</td><td>32 TCP Connections</td><td>32 TCP Connections</td></tr><tr><td>CIP™ connected messages: 1000 I/O 528⁽¹⁾</td><td>256 CIP™ connected messages</td><td>256 CIP connected messages</td><td>128 CIP connected messages</td><td>80 CIP Connected Messages</td><td>320 CIP Connected Messages</td></tr></table>	1756-EN4TR, 1756-EN4TRXT Module	1756-EN2TR, 1756-EN3TR	1756-EN2T, 1756-EN2TP, 1756-EN2TXT, 1756-EN2F Module	1756-ENBT Module	5069-AENTR, 5094-AENTxx COMPACT 5000 I/O Ethernet Adapter	5069-AEN2TR COMPACT 5000 I/O Ethernet Adapter	<ul style="list-style-type: none">50,000 pps without CIP Security25,000 pps with integrity15,000 pps with integrity and confidentiality	20,000 pps	10,000 pps	5000 pps	100,000 pps (total number of packets from both Ethernet Ports)	100,000 pps (total number of packets from both Ethernet Ports)	512 TCP connections	128 TCP connections	128 TCP connections	64 TCP connections	32 TCP Connections	32 TCP Connections	CIP™ connected messages: 1000 I/O 528 ⁽¹⁾	256 CIP™ connected messages	256 CIP connected messages	128 CIP connected messages	80 CIP Connected Messages	320 CIP Connected Messages
1756-EN4TR, 1756-EN4TRXT Module	1756-EN2TR, 1756-EN3TR	1756-EN2T, 1756-EN2TP, 1756-EN2TXT, 1756-EN2F Module	1756-ENBT Module	5069-AENTR, 5094-AENTxx COMPACT 5000 I/O Ethernet Adapter	5069-AEN2TR COMPACT 5000 I/O Ethernet Adapter																				
<ul style="list-style-type: none">50,000 pps without CIP Security25,000 pps with integrity15,000 pps with integrity and confidentiality	20,000 pps	10,000 pps	5000 pps	100,000 pps (total number of packets from both Ethernet Ports)	100,000 pps (total number of packets from both Ethernet Ports)																				
512 TCP connections	128 TCP connections	128 TCP connections	64 TCP connections	32 TCP Connections	32 TCP Connections																				
CIP™ connected messages: 1000 I/O 528 ⁽¹⁾	256 CIP™ connected messages	256 CIP connected messages	128 CIP connected messages	80 CIP Connected Messages	320 CIP Connected Messages																				

(1) There are 1000 explicit connections and 528 implicit connections.

Online Addition of Module and Connection Types

Module Type and Connection Method	In Local Chassis		Remote via an EtherNet/IP Network		Remote via a ControlNet® Network				Configure Hold Last Output State
	Offline	Runtime	Offline	Runtime	Offline		Runtime		Offline only
					Scheduled	Unscheduled	Scheduled	Unscheduled	
Digital - direct	Yes	Yes	Yes	Yes	Yes	Yes	—	Yes	Yes - 1756 I/O digital output modules
Digital - rack-optimized	—	—	Yes	Yes	Yes	—	Yes	—	Yes - 1756 I/O digital output modules
Analog - direct	Yes	Yes	Yes	Yes	Yes	Yes	—	Yes	Yes
Generic third-party - direct	Yes	Yes	Yes	Yes	Yes	Yes	—	Yes	—
1715 Redundant I/O	—	—	Yes	Yes	—	—	—	—	—
1718/1719 I/O	—	—	—	—	Yes	Yes	—	—	Yes - both analog and digital modules
1756-ENx - no connection	Yes	Yes	Yes	Yes	—	—	—	—	—
1756-ENx - rack-optimized	—	—	Yes	Yes	—	—	—	—	—
Generic EtherNet/IP third-party - direct	—	—	Yes	Yes	—	—	—	—	—
1788-EN2FFR or 1788-EN2PAR	—	—	—	—	—	—	Yes	Yes	—
1788-CN2FFR or 1788-CN2PAR	—	—	Yes	Yes	No	Yes	—	—	—
1794 FLEX I/O	—	—	Yes	—	Yes	Yes	—	—	Yes - Analog output modules only
1734 POINT I/O	—	—	Yes	—	Yes	Yes	—	—	Yes
1734 POINT Guard I/O™	Yes	—	Yes	—	—	—	—	—	—
5069 Compact 5000 I/O	Yes	—	Yes	Yes ⁽¹⁾	—	—	—	—	Yes
5069 Compact 5000 I/O Safety Modules	Yes	—	Yes	—	—	—	—	—	—
5094 FLEX 5000	—	—	Yes	Yes	—	—	—	—	Yes
5094 FLEX 5000 I/O Safety Modules	—	—	Yes	—	—	—	—	—	Yes

(1) Only supported if adding an entire rack of Compact 5000 I/O modules.

Design Considerations for Runtime/Online Addition of Modules

When you design your network, address these considerations to add modules at runtime.

Design Issue	Considerations
I/O modules	If you plan to add 1756 I/O modules at runtime, leave space in the local chassis, remote chassis on a ControlNet network, or remote chassis on an EtherNet/IP network for the I/O modules you want to add.
Other modules	You can add 1757-FFLDC devices remotely via the unscheduled portion of a ControlNet network at runtime.
Input transmission rate	Make sure the RPIs work for the data you want to send and receive. Make sure the added I/O does not depend on change of state data.
Network topology	On a ControlNet network, install spare taps so you can add modules at runtime without disrupting the network. Each tap must be terminated not to ground out the system. Check ControlNet system requirements to determine how many spare taps your network can support. <ul style="list-style-type: none"> • In a ControlNet network with redundant cabling, you can break the trunk and add a tap, but redundant cabling is lost during the module installation. • In a ControlNet ring, add a drop off the ring or add new nodes off the coax and disrupt only part of the network. • You could remove an existing node and add a repeater off that drop. Then reconnect the existing node and add any new nodes off the new segment. On an EtherNet/IP network, reserve some connection points on the switch so that you can connect additional nodes or switches in the future.
Network configuration	On a ControlNet network, plan which communication can be scheduled or unscheduled. On an EtherNet/IP network, all communication is immediate and occurs based on the module RPI (also referred to as unscheduled). If you know that you need a new chassis with digital modules in the future, configure the network and add it to the I/O configuration tree as rack optimized. Then inhibit the communication adapter until you need the chassis.
Network performance	You can add modules at runtime until you impact the capacity of the communication module. Make sure that you have sufficient communication modules for the connections you plan to add.

Determine the Appropriate Network

EtherNet/IP™, ControlNet®, and DeviceNet® networks share a universal set of communication services. These are the recommended networks for Logix control systems.

Comparison	EtherNet/IP Network	ControlNet Network	DeviceNet Network
Function	Plant management system tie-in (material handling); configuration, data collection, and control on a high-speed network	Supports transmission of time critical data between PLC processors and I/O devices	Connects low-level devices directly to plant-floor controllers—without interfacing them through I/O modules
Typical devices networked	Mainframe computers Programmable controllers Robots HMI I/O Drives Process instruments	Programmable controllers I/O chassis HMIs PCs Drives Robots	Sensors Motor starters Drives PCs Push buttons Low-end HMIs Barcode readers PLC processors Valve manifolds
Data repetition	Large packets, data sent regularly	Medium-size packets; data transmissions are deterministic and repeatable	Small packets; data is sent as needed
Number of nodes, max	Network overall: no limit	99 nodes	64 total nodes
Data transfer rate	10 Mbps, 100 Mbps, or 1000 Mbps	5 Mbps	500 Kbps, 250 Kbps, or 125 Kbps
Typical use	Plant-wide architecture High-speed applications Redundant Applications Safety Applications	Redundant applications Scheduled communication	Supply power and connectivity to low-level devices.

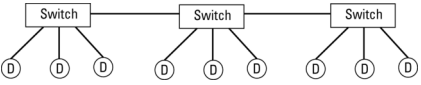
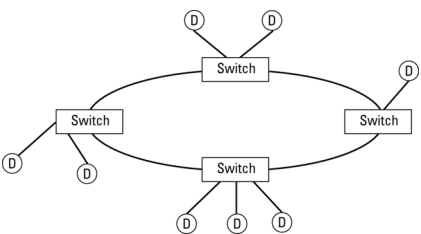
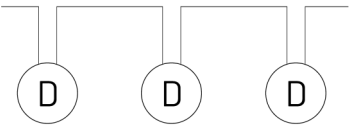
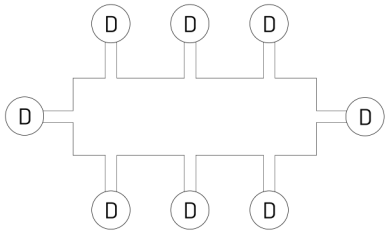
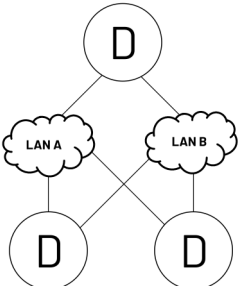
Follow these guidelines when planning a network.

Design Issue	Considerations
Network topology	Plan for future connections. Plan for additional controllers and/or communication modules to handle future I/O modules. Consider distances between devices. Determine resiliency requirements.
Network configuration	On a ControlNet network, plan which communication can be scheduled or can be unscheduled. On an EtherNet/IP network, all I/O communication is based on the RPI of the module.
Network performance	Make sure that you have sufficient communication modules for the connections you plan to use. Use available network performance tools.
Chassis	Consolidate communication connections for multiple modules to one network node. Group digital I/O modules into a rack-optimized connection to reduce the amount of communication and network bandwidth.
Input transmission rate	Make sure the RPIs work for the data you want to send and receive. Make sure that I/O added at runtime does not depend on change of state data.

EtherNet/IP Network Topology

This section features these controllers, and where applicable, the controllers are known as:

Controller Family	Includes these controllers
5580 controllers	ControlLogix® 5580 and GuardLogix® 5580 controllers
5380 controllers	CompactLogix™ 5380 and Compact GuardLogix 5380 controllers
5570 controllers	ControlLogix 5570 and GuardLogix 5570 controllers
5370 controllers	CompactLogix 5370 and Compact GuardLogix 5370 controllers

EtherNet/IP Network	Example Topologies
<ul style="list-style-type: none">• An EtherNet/IP network supports messaging, produced and consumed tags, and distributed I/O.• An EtherNet/IP network with 5570 or earlier, and 5370 or earlier controllers supports half-duplex/full-duplex, 10 Mbps or 100 Mbps operation.• An EtherNet/IP network with 5580 or 5380 controllers supports full-duplex, 10/100/1000 Mbps operation.• An EtherNet/IP network requires no network scheduling.• There are several methods available to configure EtherNet/IP network parameters for devices. Not all methods are always available. These methods are device and configuration dependent:<ul style="list-style-type: none">– DHCP– Rockwell Automation® BOOTP/DHCP utility– Programming software– Studio 5000 Logix Designer® application– RSNetWorx™ for EtherNet/IP software– Web browser– SNMP tools <p>Application Ideas</p> <ul style="list-style-type: none">• Connectivity to commercial devices (such as cameras and phones)• Business systems with remote access or sharing data• Applications with motion or safety on the same network.• Plant management (material handling)• Configuration, data collection, and control on a high-speed network• Time-critical applications with no established schedule• Inclusion of commercial technologies (such as video over IP)• Internet/Intranet connection	<p>Star</p>  <p>Ring with Switches</p>  <p>Linear</p>  <p>Device Level Ring (DLR)</p>  <p>Parallel Redundancy Protocol (PRP)</p> 

Guidelines for EtherNet/IP Networks

Guideline	Description
Use these publications.	<ul style="list-style-type: none"> EtherNet/IP Network Devices User Manual ENET-UM006A-EN-P, publication ENET-UM006 EtherNet/IP Device Level Ring Application Technique, publication ENET-AT007 EtherNet/IP Design Considerations Reference Manual, publication ENET-RM002
Data transmission depends on the controller.	<p>The type of controller determines the data transmission rate.</p> <ul style="list-style-type: none"> ControlLogix controllers transmit data at the RPI you configure for the module. CompactLogix controllers transmit data at powers of 2 ms (such as 2, 4, 8, 16, 64, or 128). For example, if you specify an RPI of 100 ms, the data actually transfers at 64 ms.
You can add I/O modules at runtime.	You can add I/O modules to remote chassis connected via an EtherNet/IP network to a running controller. You can configure direct or rack-optimized connections. For more information, see Runtime/Online Addition of Modules on page 98 .
Data transmission rate depends on the RPI.	<p>An EtherNet/IP network broadcasts I/O information to the controller based on the RPI setting. With change of state (COS) enabled and:</p> <ul style="list-style-type: none"> No data changes, the EtherNet/IP module produces data every RPI. Data changes, the EtherNet/IP module produces data at a maximum rate of RPI/4.
Select unicast EtherNet/IP communication whenever possible.	<p>To reduce bandwidth use and preserve network integrity, some facilities block multicast Ethernet packets. Multicast is a more efficient method for transmitting data with multiple consumers and redundancy applications.</p> <p>You can configure multicast or unicast connections for:</p> <ul style="list-style-type: none"> Produced and consumed tags by using the Logix Designer application I/O modules by using the Logix Designer application. <p>Unicast connections help with the following:</p> <ul style="list-style-type: none"> Let produced and consumed tag communication span multiple subnets Reduce network bandwidth. Simplify configuration for EtherNet/IP network devices because of unicast default setting for the Logix Designer application.

ControlNet Network Topology

ControlNet Network	Topology
<ul style="list-style-type: none"> A ControlNet network lets both I/O and messaging on the same wire. Multiple controllers and their respective I/O can also be placed on the same ControlNet wire. When new I/O is added, or when the communication structure on an existing I/O module changes, you must use RSNetWorx for ControlNet software to reschedule the network. If the network timing changes, every device with scheduled traffic on the network is affected. To reduce the impact of changes, place each CPU and its respective I/O on isolated ControlNet networks. Place shared I/O and produced/consumed tags on a common network available to each CPU that needs the information. Built-in redundant cabling supports I/O network and provides HMI switchover in redundant ControlLogix system. <p>Application Ideas</p> <ul style="list-style-type: none"> Default Logix network Best replacement for universal remote I/O Backbone to multiple distributed DeviceNet networks Peer interlocking network Common devices include: Logix 5000™ controllers, PanelView™ terminals, I/O modules, and drives 	<p>The diagram illustrates a ControlNet network topology. At the top, a 'Shared I/O' module is connected to a 'ControlNet Network' line. This line branches into two separate 'ControlNet Network' segments. Each segment contains a 'CPU' module and two 'I/O' modules. The 'CPU' and 'I/O' modules are connected to their respective 'ControlNet Network' lines, forming a distributed network structure.</p>

Guidelines for ControlNet Networks

Guideline	Description
Use these publications.	<ul style="list-style-type: none"> ControlNet Coax Media Planning and Installation Guide, publication CNET-IN002 ControlNet Fiber Media Planning and Installation Guide, publication CNET-IN001 ControlNet Network Configuration User Manual, publication CNET-UM001
Adjust the default RSNetWorx for ControlNet settings.	<p>Change these settings in the RSNetWorx for ControlNet software:</p> <ul style="list-style-type: none"> UMAX (highest unscheduled node on the network) <ul style="list-style-type: none"> Default is 99 The network takes the time to process the total number of nodes that are specified in this setting, even if there are not that many devices on the network Change to a reasonable level to accommodate the active network devices and additional devices that can be connected SMAX (highest scheduled node on the network) <ul style="list-style-type: none"> Default is 1 This must be changed for all systems Set SMAX < UMAX
Design for at least 400 KB of available, unscheduled network bandwidth, as displayed by RSNetWorx for ControlNet software.	<p>Leaving too little bandwidth for unscheduled network communication results in poor message throughput and slower workstation response.</p> <p>Unscheduled data transfers on ControlNet occur asynchronous to the program scan and support a maximum of 510 bytes/node per ControlNet NUT.</p>
Place DeviceNet (1756-DNB) communication modules in the local chassis.	<p>DeviceNet (1756-DNB) communication modules have multiple, 500-byte data packets that impact scheduled bandwidth. Place these modules in the same chassis as the controller to avoid this data being scheduled over the DeviceNet network.</p> <p>If you must place these communication devices in remote chassis, configure the input and output sizes to match the data that is configured in RSNetWorx for DeviceNet software. This reduces the amount of data that must be transmitted.</p>
Limit 1756-CNB, 1756-CNBR connections.	<p>For best performance, limit the 1756-CNB, 1756-CNBR to 40...48 connections. Add additional 1756-CNB, 1756-CNBR modules in the same chassis if you need more connections. To improve system performance, you can add more modules and split connections among the modules.</p> <p>If the chassis that contains the CNB module also contains multiple digital I/O modules, configure the CNB communication format for Rack Optimization. Otherwise, use None.</p> <p>As a cost savings measure, use 1756-CNB, 1756-CNBR modules in chassis that contain only I/O modules for traditional adapter functionality. Use the 1756-CN2, 1756-CN2R, 1756-CN2RXT modules in the same chassis as the controller for traditional scanner functionality.</p>
For additional connections, consider the 1756-CN2, 1756-CN2R, 1756-CN2RXT modules.	<p>The 1756-CN2/B, 1756-CN2R/B, 1756-CN2RXT communication modules each support 131 connections, and have higher performance than previous modules.</p> <p>The 1756-CN2/A, 1756-CN2R/A communication modules each support 100 connections.</p>
If you change network settings, resave each controller project.	<p>Any time that you edit the network with RSNetWorx for ControlNet software and you save or merge your edits, connect to each controller in the system with their respective project file and perform a save and upload. This copies the ControlNet settings into the offline, database file and makes sure that future downloads of the controller permit it to go online without having to run RSNetWorx for ControlNet software.</p>
You can add I/O modules at runtime.	<p>You can add 1756 I/O modules and some drives to remote chassis connected via ControlNet to a running controller. It is recommended to use a 1756-CN2/B, 1756-CN2R/B, or 1756-CN2RXT module as the traditional scanner in these applications.</p>
Data transmission depends on the controller.	<p>The type of controller determines the data transmission rate.</p> <ul style="list-style-type: none"> ControlLogix controllers transmit data at the RPI you configure for the module. CompactLogix controllers transmit data at powers of 2 ms (such as 2, 4, 8, 16, 64, and 128). For example, if you specify an RPI of 100 ms, the data actually transfers at 64 ms.

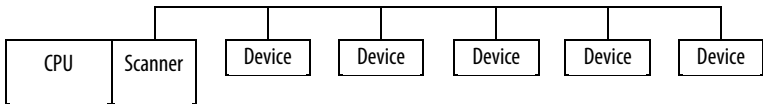
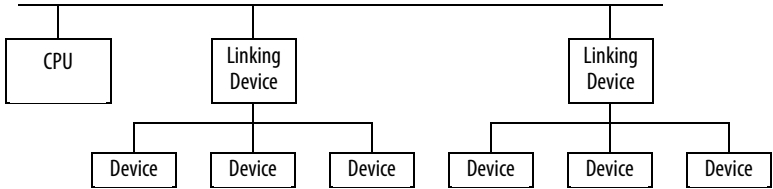
Guidelines for Unscheduled ControlNet Networks

Guideline	Description																										
You can run an entire ControlNet network as unscheduled.	<p>An unscheduled ControlNet network:</p> <ul style="list-style-type: none"> provides for easier network configuration. is useful if your I/O updates needs are slower. supports the addition of 1756 I/O modules and some drives without placing the controller in Program mode. provides an HMI network with fast switchover times in a redundant controller system. <p>You must still run RSNetWorx for ControlNet software at least once to configure NUT, SMAX, UMAX, and media configuration settings.</p>																										
Plan appropriately if you place I/O on an unscheduled ControlNet network.	<p>Follows these recommendations for I/O on an unscheduled ControlNet network:</p> <ul style="list-style-type: none"> A 1756-CN2, 1756-CN2R Series B or later, or a 1756-CN2RXT is recommended. Disable the Change of State (COS) feature on digital input modules because it can cause inputs to be sent faster than the RPI. Set the real-time sample (RTS) on analog cards slower than the RPI Dedicate a ControlNet network to I/O only. Do not exceed 80% utilization of a 1756-CN2, 1756-CN2R, 1756-CN2RXT communication module. Do not exceed 75% utilization of a 1756-CNB, 1756-CNBR communication module. Have no more than 48 connections on the 1756-CNB, 1756-CNBR communication module. Use a NUT of 10 ms or more. Keep the SMAX and UMAX values as small as possible. 																										
1756-CNB, 1756-CNBR only Set the RPI at 25 ms or slower.	<p>Use RPI of 25 ms or slower for unscheduled modules to avoid overload on the 1756-CNB, 1756-CNBR communication module. Depending on the RPI, the communication module loading increases 1...4% for each I/O module added.</p> <p style="text-align: center;">Additional 1756-CNB, 1756-CNBR Loading</p> <table border="1"> <caption>Approximate data for Additional 1756-CNB, 1756-CNBR Loading</caption> <thead> <tr> <th>RPI (ms)</th> <th>Additional CNB Loading %</th> </tr> </thead> <tbody> <tr><td>25</td><td>4.0</td></tr> <tr><td>45</td><td>2.5</td></tr> <tr><td>65</td><td>1.8</td></tr> <tr><td>85</td><td>1.4</td></tr> <tr><td>105</td><td>1.1</td></tr> <tr><td>125</td><td>0.9</td></tr> <tr><td>145</td><td>0.8</td></tr> <tr><td>165</td><td>0.7</td></tr> <tr><td>185</td><td>0.6</td></tr> <tr><td>205</td><td>0.55</td></tr> <tr><td>225</td><td>0.52</td></tr> <tr><td>245</td><td>0.5</td></tr> </tbody> </table>	RPI (ms)	Additional CNB Loading %	25	4.0	45	2.5	65	1.8	85	1.4	105	1.1	125	0.9	145	0.8	165	0.7	185	0.6	205	0.55	225	0.52	245	0.5
RPI (ms)	Additional CNB Loading %																										
25	4.0																										
45	2.5																										
65	1.8																										
85	1.4																										
105	1.1																										
125	0.9																										
145	0.8																										
165	0.7																										
185	0.6																										
205	0.55																										
225	0.52																										
245	0.5																										
1756-CNB, 1756-CNBR only The RPI affects how many I/O modules you can have.	<p>This chart shows the number of modules and associated RPIs so that you do not exceed 75% utilization of the 1756-CNB, 1756-CNBR communication module.</p> <p style="text-align: center;">Maximum Number of I/O Modules in an Unscheduled Network</p> <table border="1"> <caption>Approximate data for Maximum Number of I/O Modules in an Unscheduled Network</caption> <thead> <tr> <th>RPI (ms)</th> <th>Number of Unscheduled Modules</th> </tr> </thead> <tbody> <tr><td>25</td><td>15</td></tr> <tr><td>30</td><td>18</td></tr> <tr><td>35</td><td>21</td></tr> <tr><td>40</td><td>24</td></tr> <tr><td>45</td><td>27</td></tr> <tr><td>50</td><td>30</td></tr> <tr><td>55</td><td>33</td></tr> <tr><td>60</td><td>36</td></tr> <tr><td>65</td><td>39</td></tr> <tr><td>70</td><td>42</td></tr> <tr><td>75</td><td>45</td></tr> <tr><td>80</td><td>48</td></tr> </tbody> </table>	RPI (ms)	Number of Unscheduled Modules	25	15	30	18	35	21	40	24	45	27	50	30	55	33	60	36	65	39	70	42	75	45	80	48
RPI (ms)	Number of Unscheduled Modules																										
25	15																										
30	18																										
35	21																										
40	24																										
45	27																										
50	30																										
55	33																										
60	36																										
65	39																										
70	42																										
75	45																										
80	48																										

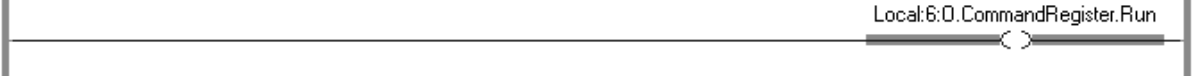
Compare Scheduled and Unscheduled ControlNet Communication

Scheduled ControlNet Communication	Unscheduled ControlNet Communication
Deterministic	Less deterministic than scheduled communication Provides simpler ControlNet installations when scheduled networks are not required
To add scheduled I/O on the ControlNet network, you must: <ul style="list-style-type: none"> Add the I/O to an offline controller project. Download the project to the controller. Run RSNetWorx to schedule the network requires network to be scheduled (must stop the network and put the controller in Program mode to schedule a network). Save the controller project. 	Can be changed online without impacting the schedule New modules can affect other modules communicating via unscheduled bandwidth Supports 1756 I/O modules and some drives
RPI and NUT determine module communication rates	RPI determines module communication rates
MSG and HMI traffic can occur on the same network because they are isolated in unscheduled traffic MSG and HMI traffic do not affect I/O communication	Recommend 1756-CN2, 1756-CN2R, 1756-CN2RXT Recommend a dedicate ControlNet network for only I/O modules MSG and HMI traffic can affect I/O communication
Direct and rack-optimized connections to I/O	Only direct connections to I/O (results in being able to use fewer total I/O modules because of the connection limits of controllers and communication modules)
Supports any firmware revision of a ControlNet communication module	You can use any 1756-CN2, 1756-CN2R, 1756-CN2RXT communication module If you use a 1756-CNB, 1756-CNBR communication module, it must be series D or later
Supports any I/O platform that can communicate via a ControlNet network	Supports only 1756 I/O modules

DeviceNet Network Topology

DeviceNet Network	Topology
<ul style="list-style-type: none"> You need a DeviceNet scanner to connect the controller to DeviceNet devices. You must use RSNetWorx for DeviceNet software to configure devices and create the scanlist for the scanner. You can configure the network communication rate as 125 Kbps (default and a good starting point), 250 Kbps, or 500 Kbps. If each device on the network (except the scanner) sends ≤ 4 bytes of input data and receives ≤ 4 bytes of output data, you can use AutoScan to configure the network. <p>Application Ideas</p> <ul style="list-style-type: none"> Distributed devices Drives network Diagnostic information 	<p>Single Network</p>  <pre> graph TD CPU[CPU] --- Bus1[] Scanner[Scanner] --- Bus1 Bus1 --- Device1[Device] Bus1 --- Device2[Device] Bus1 --- Device3[Device] Bus1 --- Device4[Device] Bus1 --- Device5[Device] </pre> <p>Several Smaller Distributed Networks (Subnets)</p>  <pre> graph TD CPU[CPU] --- Bus2[] LD1[Linking Device] --- Bus2 LD2[Linking Device] --- Bus2 Bus2 --- Bus3[] Bus3 --- Device1[Device] Bus3 --- Device2[Device] Bus3 --- Device3[Device] Bus2 --- Bus4[] Bus4 --- Device4[Device] Bus4 --- Device5[Device] Bus4 --- Device6[Device] </pre>

Guidelines for DeviceNet Networks

Guideline	Description
Use these publications.	<ul style="list-style-type: none"> DeviceNet Cable System Manual, publication DNET-UM072 DeviceNet Network Configuration User Manual, publication DNET-UM004
Use the DeviceNet Tag Generator tool.	<p>The Logix Designer application includes a DeviceNet tag generator tool that creates device-specific structured tags and logic based on the network configuration in RSNetWorx for DeviceNet software.</p> <p>The logic copies data to and from the DNB data array tags to the device tags so that data is presented synchronously to program scan.</p>
Place DeviceNet (DNB) communication modules in the local chassis.	<p>Place DNB modules in the local chassis to help maximize performance, especially in ControlLogix systems.</p> <p>Size the input and output image for the DNB modules to the actual devices that are connected plus 20% for future growth. If you have to place DNB modules in remote chassis, sizing the input and output images is critical for best performance.</p>
Verify that the total network data does not exceed the maximum DNB data table size.	<p>A DNB supports:</p> <ul style="list-style-type: none"> 124, 32-bit input words. 123, 32-bit output words. 32, 32-bit status words. <p>You can use RSNetWorx for DeviceNet software offline to estimate network data. Use a second DNB if there is more network data than one module can support.</p>
Configure slaves first.	<p>Configure device parameters before adding that device to the scanlist. You cannot change the configuration of many devices once they are already in the scanlist.</p> <p>If you configure the scanner first, there is a chance that the scanner configuration cannot match the current configuration for a device. If the configuration does not match, the device does not show up when you browse the network.</p>
Leave node address 63 open to add nodes.	<p>Devices default to node 63 out-of-the-box. Leave node address 63 unused so you can add a new device to the network. Then change the address of the new device.</p>
Leave node address 62 open to connect a computer.	<p>Always leave at least one open node number to let a computer be attached to the network if needed for troubleshooting or configuration.</p>
Don't forget to set the scanner run bit.	<p>For the scanner to be in Run mode, the controller must be in Run mode and the logic in the controller must set the scanner run bit.</p>
	
Make sure that you have the most current EDS files for your devices.	<p>RSNetWorx for DeviceNet software uses EDS file to recognize devices. If the software is not properly recognizing a device, you are missing the correct EDS files. For some devices, you can create an EDS file by uploading information from the device. Or you can get EDS files from: http://www.ab.com/networks/eds.</p>

Notes:

Communicate with Other Devices

The MSG instruction asynchronously reads or writes a block of data to another device.

If the target device is a	Select one of these message types
Logix 5000™ controller	CIP™ Data Table Read
	CIP Data Table Write
I/O module that you configure with the Studio 5000 Logix Designer® application	Module Reconfigure
	CIP Generic
SERCOS drive	SERCOS IDN Read
	SERCOS IDN Write
PLC-5® controller	PLC5 Typed Read
	PLC5 Typed Write
	PLC5 Word Range Read
	PLC5 Word Range Write
SLC™ controller MicroLogix™ controller	SLC Typed Read
	SLC Typed Write
Block transfer module	Block Transfer Read
	Block Transfer Write
PLC-3® processor	PLC3 typed read
	PLC3 typed write
	PLC3 word range read
	PLC3 word range write
PLC-2® processor	PLC2 unprotected read
	PLC2 unprotected write

Cache Messages

Some types of messages use a connection to send or receive data. Some also give you the option to either leave the connection open (cache) or close the connection when the message is done transmitting. This table shows messages that use a connection and whether you can cache the connection.

Message Type	Communication Method	Uses Connection	Can Cache Connection
CIP data table read or write	CIP	Yes	Yes
PLC2, PLC3, PLC5, or SLC (all types)	CIP		
	CIP with Source ID		
	DH+™	Yes	Yes
CIP generic	N/A	Your option ⁽¹⁾	Your option ⁽¹⁾
Block transfer read or write	N/A	Yes	Yes

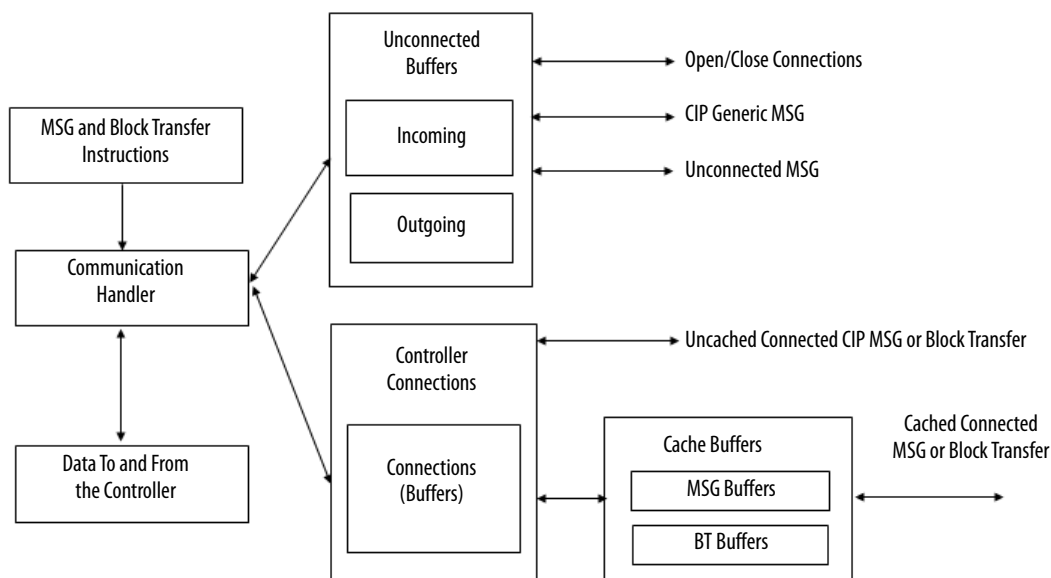
(1) You can connect CIP generic messages, but for most applications we recommend that you leave CIP generic messages unconnected.

A cached connection remains open until one of the following occurs:

- The controller goes to Program mode.
- You rerun the message as uncached.
- Another message is initiated and a cached buffer is needed.
- An intermediate node in the connection goes down.

Message Buffers

The controller has buffers for unconnected messages and for cached messages. Buffers store incoming and outgoing message data until the controller can process the data.



Buffer	Description
Outgoing, connected	<p>The outgoing unconnected buffers are for:</p> <ul style="list-style-type: none"> Establishing I/O connections to local I/O modules and remote devices on ControlNet®, EtherNet/IP™, and remote I/O networks. Executing unconnected PLC2, PLC3, PLC5, or SLC (all types) messages over Ethernet/IP or ControlNet (CIP and CIP with Source ID) networks. Initiation of messaging over a DH+™ network (uses 2 buffers, one to open the connection and one to transfer data). Initiation of uncached block transfers. Initiation of uncached CIP read/write message instructions. Initiation of cached block transfers. Initiation of cached CIP read/write messages instructions. CIP Generic message instructions.
Incoming, unconnected	<p>The incoming unconnected buffers:</p> <ul style="list-style-type: none"> Initially receive a cached CIP message instruction. Receive an uncached CIP message instruction. Receive a message over a DH+ network. Receive a CIP Generic message instruction. Receive a read or write request from a ControlNet PanelView™ terminal (unconnected messaging). Initially receive of a read request from an EtherNet/IP PanelView terminal (connected messaging). Receive a write request from an EtherNet/IP PanelView terminal (unconnected messaging). Receive an initial request from the Logix Designer application to go online. Initially receive Linx-based connections.
Cached buffers	<p>The cached buffers are outgoing buffers for messages and block transfers. A cached connection helps message performance because the connection is left open and does not need to be re-established the next time that it is executed. A cached connection counts towards the total limit of connections for a controller. A cached connection is refreshed at the connection RPI. All cached entries are closed when the controller transitions to Program mode. The first time a cached message is executed, it uses one of the outgoing unconnected buffers. When the connection is established, it moves into the cached buffer area.</p> <p>For optimum performance, do not cache more messages or block transfers than there are cached buffers. If you cache more than the available cached buffers, the controller looks for a connection that has been inactive for the longest time, closes that connection, and lets a new connection take its place. The controller closes a cached message or block transfer, depending on which has been inactive the longest. If all cached connections are in use, the message is executed as connected and, once it is completed, the connection is closed.</p> <p>You can multiplex cached connections. If a connection is inactive and a message instruction executes that has the same target and path, it uses that inactive connection. For example, if you have a block transfer read and write to the same module, interlock the read and write so that only one is active at a time. Then when they are cached, they use the same cached connection.</p>

Outgoing Unconnected Buffers

IMPORTANT This section does not apply to 5380, 5480, or 5580 controllers.

Buffers	Use
1...10	The first 10 buffers (default) are shared for unconnected messaging, initiating connected messaging, establishing I/O connections, and establishing produced/consumed connections.
11	The 11th buffer is dedicated to establishing I/O and produced/consumed connections.
12...40	The 12th to the 40th buffers are used only for initiating connected messages and executing unconnected messages. To increase the outgoing buffers to a value higher than 11, execute a CIP generic message to configure that change each time you transition from Program mode to Run mode.

Guidelines for Messages

Guideline	Description
Message tags may be created at controller scope or program scope.	The operating system accesses the information in a message tag asynchronously to the program scan. Along with the visible fields within the message tag, there are hidden attributes that are only referenced by the background operating system.
You can use a message to send a large amount of data.	Even though there are network packet limitations (such as 500 bytes on ControlNet and 244 bytes on DH+), the controller can send a large amount of data from one MSG instruction. When configuring the message, select an array as the source/destination tags and select the number of elements (as many as 32,767 elements) you want to send. The controller automatically breaks the array into small fragments and sends all of the fragments to the destination. On the receiving side, the data appears in fragments, so some application code can be required to detect the arrival of the last piece.
Do not manipulate the message status bit	<p>Do not change the following status bits of a MSG instruction:</p> <ul style="list-style-type: none"> • DN • EN • ER • EW • ST <p>Do not change those bits either by themselves or as part of the FLAGS word. If you do, the controller can have a nonrecoverable fault. The controller clears the project from its memory when it has a nonrecoverable fault.</p>

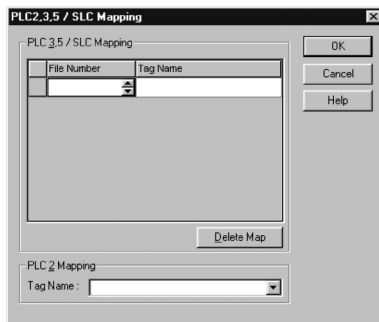
Guidelines to Manage Message Connections

Guideline	Description
Create user-defined structures or arrays.	<p>User-defined structures let you organize your data to match your machine or process.</p> <ul style="list-style-type: none"> • One tag contains all data that is related to a specific aspect of your system. This keeps related data together and easy to locate, regardless of its data type. • Each individual piece of data (member) gets a descriptive name. This automatically creates an initial level of documentation for your logic. • You can use the structure to create multiple tags with the same data layout. • Linux-based software can optimize user-defined structures more than standalone tags.
Cache connections when appropriate.	<p>If a message executes repeatedly, cache the connection. This keeps the connection open and optimizes execution time. You can increase execution time if you open a connection each time the message executes.</p> <p>If a message executes infrequently, do not cache the connection. This closes the connection upon completion of the message, which frees up that connection for other uses.</p>

Guidelines for Block Transfer Messages

Guideline	Description
Distribute 1771 analog modules across multiple chassis.	To reduce the number of block transfers that one 1771-ACN or 1771-ASB module manages, distribute 1771 analog modules across multiple chassis.
Isolate different 1771 chassis on different networks.	When you isolate different chassis onto different networks, it diversifies the communication so that no single network or communication module has to deal with all communication.
Increase ControlNet unscheduled bandwidth.	If you communicate over a ControlNet network, increase the amount of ControlNet unscheduled bandwidth to permit additional time on the network for data exchange. See Compare Scheduled and Unscheduled ControlNet Communication on page 106 for more information about unscheduled bandwidth on a ControlNet network.
Increase the system overhead timeslice percentage on ControlLogix® 5570 or earlier and CompactLogix™ 5370 or earlier controllers.	Increase the system overhead timeslice to allocate more CPU time to communication processing from the continuous task.
Interlock block transfer read and write messages to the same module.	Programmatically interlock block transfer read and write messages to the same module so that both operations cannot be active simultaneously.
Use the 1756-RIO module for systems with a high number of block transfer modules.	The 1756-RIO module provides connectivity from a ControlLogix chassis to 1771 I/O and other modules that are connected via remote I/O. The 1756-RIO module off-loads the burden of performing block transfers from the controller and increases the number of block transfer operations that can be performed.

Map Tags



A Logix 5000™ controller stores tag names on the controller so that other devices can read or write data without having to know physical memory locations. Many products only understand PLC/SLC data tables, so the Logix 5000 controller offers a PLC/SLC mapping function that lets you map Logix tag names to memory locations.

- You only have to map the file numbers that are used in messages; the other file numbers do not need to be mapped.
- The mapping table is loaded into the controller and is used whenever a logical address accesses data.
- You can only access controller-scoped tags (global data).

Follow these guidelines when you map tags.

- Do not use file numbers 0, 1, and 2. These files are reserved for Output, Input, and Status files in a PLC-5 processor.
- Use PLC-5 mapping only for tag arrays of data type INT, DINT, or REAL. Attempting to map elements of system structures can produce undesirable effects.
- Use these file types and identifiers.

For this Logix 5000 array type	Use this PLC file identifier
INT array	N or B
DINT array	L
REAL array	F

Notes:

Alarms

The following options are available for alarms:

Alarm Option	Description
FactoryTalk® Alarms and Events	<p>The FactoryTalk Alarms and Events system integrates alarming between FactoryTalk View SE applications and the controllers by embedding an alarming engine in the controllers.</p> <ul style="list-style-type: none"> • Uniform, micro-second accurate, time stamps determined at the alarm source • Server-client model so all clients get the updates simultaneously • Single connection to controller • Combines all alarms in the system into one uniform view • Supports ControlLogix®, CompactLogix™, SLC™, PLC-5®, and non-Rockwell Automation controller <p>For more information, see FactoryTalk Alarms and Events Configuration Guide, FTAE-RM001.</p>
Controller tag-based alarms	<p>Tag-based alarms monitor a tag value to determine the alarm condition. Tag-based alarms are not part of the logic program and do not increase the scan time for a project.</p> <ul style="list-style-type: none"> • Alarms defined on tags with periodic evaluation • Standards-based design • Leverages existing FactoryTalk Alarms and Events infrastructure <p>Only 5380 and 5580 controllers support tag-based alarms.</p>
Controller instruction-based alarms	<p>Instruction-based alarms are generated and maintained by using an instruction in a Logix controller. Two Logix-based alarm instructions that are available in relay ladder, structured text, and function block diagram.</p> <ul style="list-style-type: none"> • The Digital Alarm (ALMD) instruction detects alarms that are based on Boolean (true/false) conditions. • The Analog Alarm (ALMA) instruction detects alarms that are based on the level or rate of change of analog values. <p>Alarm instructions and their specific data types consume a larger portion of controller memory and scan time than tag-based alarms.</p> <ul style="list-style-type: none"> • Alarms are detected at the same time logic is executed • Alarms events are buffered in controller memory • Alarms events are pushed to the HMI only on state changes

Guidelines for Tag-Based Alarms

Tag-based alarms are useful when you want to maximize controller scan time and when you want to integrate alarms with legacy or third-party devices.

Guideline	Description
An alarm definition is associated with an Add-On Instruction (AOI) or a defined data type.	<p>When a tag is created using a data type or an AOI that has alarm definitions, alarms are created automatically based on the alarm definitions.</p> <p>You can create an alarm definition for the following components:</p> <ul style="list-style-type: none"> • Tag or parameter of an AOI. • Member of a user-defined data type (UDT) • Member of a system-defined data type • Member of a module-defined data type <p>When a tag uses a data type that has alarm definitions that are associated with it, alarm conditions are automatically added for the tag based on its alarm definitions.</p> <p>When an AOI is based on an AOI definition, alarm conditions are automatically added for the AOI instance based on the alarm definitions that are associated with the AOI definition.</p>
Use the keyword THIS to create modular logic to access alarm attributes	<p>When the logic executes, the controller replaces THIS with the name of a program, controller, or Add-On Instruction. The keyword THIS lets you create a logic module that can be inserted in any project on any controller and still execute correctly. Use ::THIS to represent a controller name, \THIS to represent a program name, and THIS to represent the name of an Add-On Instruction.</p>
Access individual alarm attributes	<p>Use tag-based alarm attributes as operands in instructions to access tag-based alarm attributes or attributes from a set of alarms. For example, the alarm set for a controller, program, or for instances of an Add-On Instruction.</p>

Guideline	Description
Access alarm set attributes	Use alarm set attributes as operands in instructions to access the attributes.
Access individual alarm attributes in Add-On Instruction definitions	Use individual alarm attributes as operands in Add-On Instruction (AOI) definitions to access the attributes of alarms that are associated with local tags, input parameters, or output parameters within the same AOI instance.
Access attributes from Add-On Instruction alarm sets	The alarms contained in an Add-On Instruction (AOI) definition, a structured tag of an AOI definition, or an array tag of an AOI definition can be referenced as an alarm set. Use these alarm set attributes as operands in logic. When you reference an attribute from an individual alarm, you insert the owner of the alarm in the operand syntax. Similarly, when you reference an attribute from an AOI alarm set, you insert the alarm set container (the AOI definition, AOI structured tag, or AOI array tag) in the operand syntax.

Access Tag-based Alarms

Instructions can access the members of the following tag-based alarms:

- Alarm conditions that are associated with controller tags.
- Alarm conditions that are associated with local scalar tags, input parameters, output parameters, or public parameters within the same program.
- Alarm conditions that are associated with input, output, or public parameters of other programs.
- Alarm definitions that are associated with local tags, input parameters, or output parameters of an Add-On Instruction.
- Alarm definitions that are associated with local tags, input parameters, or output parameters of nested Add-On Instructions.

Instructions can access the following alarm sets or members of alarm sets:

- Alarm sets that are associated with the controller or with a program.
- Alarm sets that are associated with controller tags.
- Alarm sets that are associated with local tags, input parameters, output parameters, or public parameters within the same program.
- Alarm sets that are associated with local tags, input parameters, output parameters, or public parameters of other programs.
- Alarm sets that are associated with input, output, or public parameters in other programs.
- Alarm sets that are associated with local tags, input parameters, or output parameters of an Add-On Instruction.
- An alarm condition or an alarm set of an input, output, or public parameter in another program.
- Alarm sets that are associated with local tags, input parameters, or output parameters of nested Add-On Instructions.

Instruction cannot access the members of the following tag-based alarms:

- Alarm conditions that are associated with local tags in other programs.
- Alarm definitions that are associated with the parent Add-On Instruction.
- Alarm definitions that are associated with tags or parameters of the parent Add-On Instruction.

Instruction cannot access the members of the following alarm sets:

- Alarm sets that are associated with local tags in other programs.
- Alarm sets that are associated with the parent Add-On Instruction.
- Alarm sets that are associated with tags or parameters of the parent Add-On Instruction.

Instructions cannot access the following alarm attributes:

- Evaluation period, alarm condition types, and expressions.
- Time-related attributes that have LINT data types.
- Attributes that have STRING data types.
- Reference type attributes, such as associate tags, target tags, and input tags.

Guidelines for Instruction-Based Alarms

Instruction-based alarms are useful when you want timestamps on alarm or you want to integrate alarms without modifying the HMI displays.

Guideline	Description
Estimate increased controller memory use for each alarm.	<p>The alarm instructions use new alarm data types that contain state information and time stamps for each alarm. Estimate this memory use in the controller:</p> <ul style="list-style-type: none"> • 2 KB per FactoryTalk Alarms and Events subscriber that receives alarms from the controller • There is a maximum of 16 subscribers per controller. Most applications only require one subscriber to a controller to provide data to many FactoryTalk View SE clients. • 2.2 KB per alarm (typical), depends on use of associated tags
Alarm instructions increase total controller scan time.	<p>The ALMD instructions and ALMA instructions affect total scan time. See Logix 5000™ Controllers Instruction Execution Time and Memory Use Reference Manual, publication 1756-RM087 for execution times for your controller firmware.</p> <p>An alarm state change is any event that changes the condition of the alarm, such as acknowledgment or suppression of the alarm. Creating dependencies on related alarms to minimize the potential for many alarms to change state simultaneously (alarm bursts). Large alarm bursts can have a significant impact on application code scan time.</p> <p>Important: In redundancy systems, consider scan time impact due to crossloading alarm tag data. For more information see the ControlLogix Enhanced Redundancy System User Manual, publication 1756-UM535.</p>
You can edit or add an alarm instruction online.	<p>Online edits of new and existing alarms are automatically sent to the subscribers. You do not have to re-subscribe to receive the updated information. Online changes automatically propagate from the controller alarm structure to the rest of the architecture.</p>
In relay ladder, how you define the alarm values on the instruction determines whether you can access those values programmatically through the alarm structure.	<p>When you create an alarm instruction, you also create an alarm data type for that alarm. For example, MyDigitalAlarm of data type DigitalAlarm. In relay ladder, the following values are shown on the instruction:</p> <ul style="list-style-type: none"> • ProgAck • ProgReset • ProgDisable • ProgEnable <p>If you enter a value or assign a tag to these faceplate parameters (such as AckSection1All), the value or tag value is automatically written to the alarm structure each time the instruction is scanned.</p> <p>If you want to programmatically access the alarm structure, you must assign the structure tag to the faceplate. For example, to use MyAnalogAlarm.ProgAck in logic, assign the tag MyAnalogAlarm.ProgAck on the faceplate to the ProgAck parameter.</p>
Test alarm behavior from within the Logix Designer application.	<p>On the Status tab of the alarm dialog, monitor the alarm condition, acknowledge an alarm, disable an alarm, suppress an alarm, or reset an alarm. Use the dialog selections to see how an alarm behaves, without needing an operational HMI.</p>

Guideline	Description
Reduce mistakes by making sure that alarms are noticed.	Shelving an alarm removes the alarm from the operator view for a period of time. It is like suppressing an alarm, except that shelving is time-limited. If an alarm is acknowledged while it is shelved, it remains acknowledged even if it becomes active again. It becomes unacknowledged when the shelf duration ends provided the alarm is still active at that moment.
Increase productivity by eliminating nuisance alarms.	Set a duration (ms) on the ALMA instruction to specify how long an alarm condition must exist before being reported. Apply the duration to individual, analog alarm levels.
High availability of alarm data helps reduce material losses.	Previous to revision 31, the alarm log in the controller stores the last 10,000 alarm state transitions in a circular log. This replaces the alarm buffer in controllers with firmware earlier than revision 21.

Configure Logix-based Alarm Instructions

Option	Description
Message string	<p>The message string (maximum of 255 characters, including embedded text) contains the information to display to the operator regarding the alarm. Besides entering text, you can also embed variable information. In the alarm message editor, select the variable that you want and add it anywhere in the message string.</p> <p>You cannot programmatically access the alarm message string from the alarm tag. To change the alarm message based on specific events, configure one of the associated tags as a string data type and embed that associated tag in the message.</p> <p>You can maintain alarm messages in multiple languages. Either enter the different languages in the associated language versions of the Logix Designer application or in an import/export (.CSV or .TXT) file.</p>
Associated tags	<p>You can select as many as four additional tags from the controller project to associate with the alarm. These tags are sent with an alarm message to the alarm server. Associated tags can be BOOL, INT, SINT, DINT, REAL, or string data types. For example, a digital alarm for a pressure relief valve can also include information such as pump speed and system pressure, and tank temperature.</p> <p>Optionally, embed the associated tags into the message text string.</p>
Severity	<p>Use the configurable severity range from 1...1000 to rank the importance of an alarm. A severity of 1 is for low-priority alarms; a severity of 1000 is for an emergency condition.</p> <p>You can configure how the FactoryTalk ranges are presented to the operator. The operator can also filter on alarm levels. For example, a maintenance engineer can filter to see only those alarms at severity 128.</p>
Alarm class	<p>Use the alarm class to group related alarms. Specify the alarm class the same for each alarm you want in the same class. The alarm class is case-sensitive.</p> <p>For example, specify class Control Loop to group all alarms for PID loops.</p> <p>You can then display and filter alarms at the HMI based on the class. For example, an operator can display all tank alarms or all PID loop alarms.</p> <p>The alarm class does not replace subscription to specific alarms. The FactoryTalk View SE Alarm object graphics have configuration options to determine which controller alarms an operator sees.</p>
View command	Execute a command on the operator station when requested by an operator for a specific alarm. This lets an operator execute any standard FactoryTalk View command, such as call specific faceplates and displays, execute macros, access help files, and launch external applications. When the alarm condition occurs and is displayed to the operator, a button on the summary and banner displays lets the operator run an associated view command.
Defaults	The Parameters tab of the alarm instruction properties lets you define values for instruction parameters. You can return the parameters to factory defaults and you can define your own set of instruction defaults. The instruction defaults you assign are defaults for only that instance of the instruction.

Automatic Diagnostics

Automatic Diagnostics provide device diagnostics to HMIs and other clients, with zero programming. You need:

- ControlLogix 5580, GuardLogix 5580, CompactLogix 5380, Compact GuardLogix 5380, or CompactLogix 5480 controller with firmware revision 33 or later.
- FactoryTalk View SE version 12 or later to add the Automatic Diagnostic object to FactoryTalk View displays. See the FactoryTalk View Site Edition User's Guide, publication [VIEWSE-UM006](#).
- FactoryTalk Alarms and Events version 6.20 or later to subscribe to the diagnostic messages created in the controller. See the FactoryTalk Alarms and Events System Configuration Guide, publication [FTA-E-RM001](#).

The diagnostics include device description conditions and state events. Diagnostics based on the device definition (such as fault or open wire) are sent to a compatible Rockwell Automation HMI device or software, and displayed on the Automatic Diagnostic Event Summary object.

Advanced Diagnostics v33 Demo

Automatic Diagnostics v33 Demo

No filter

State	Ass...	Event Time	Area	Device Name	Catalog	Product Type	Message
✖		4/13/2020 10:16:17 AM	[AD_Demo]Remote_DLR	1756-EN2TR	Communications Adapter	Connection Lost with Device	
✖		4/13/2020 10:16:22 AM	[AD_Demo]OB16D	1756-OB16D/A	General Purpose Discrete I/O	Point 1, No Load - The wire is disconnected from the module.	
✖		4/13/2020 10:16:22 AM	[AD_Demo]OB16D	1756-OB16D/A	General Purpose Discrete I/O	Point 2, No Load - The wire is disconnected from the module.	
✖		4/13/2020 10:16:22 AM	[AD_Demo]OB16D	1756-OB16D/A	General Purpose Discrete I/O	Point 4, No Load - The wire is disconnected from the module.	
✖		4/13/2020 10:16:22 AM	[AD_Demo]OB16D	1756-OB16D/A	General Purpose Discrete I/O	Point 9, No Load - The wire is disconnected from the module.	
✖		4/13/2020 10:16:22 AM	[AD_Demo]OB16D	1756-OB16D/A	General Purpose Discrete I/O	Point 10, No Load - The wire is disconnected from the module.	
✖		4/13/2020 10:16:22 AM	[AD_Demo]OB16D	1756-OB16D/A	General Purpose Discrete I/O	Point 12, No Load - The wire is disconnected from the module.	
✖		4/13/2020 10:16:22 AM	[AD_Demo]OB16D	1756-OB16D/A	General Purpose Discrete I/O	Point 14, No Load - The wire is disconnected from the module.	
✖		4/13/2020 10:16:22 AM	[AD_Demo]OB16D	1756-OB16D/A	General Purpose Discrete I/O	Point 15, No Load - The wire is disconnected from the module.	
✖		4/13/2020 10:21:36 AM	[AD_Demo]AutomaticDiagnostics_v33_Demo	1756-L85E	Programmable Logic Controller	Major Fault T04:C34 - Program Fault: A timer instruction had a negative value for its PRE or ACC.	

State: Active and Unsuppressed
 Assessment: FAILED
 Event Time: 4/13/2020 10:21:36 AM
 Area:
 Device Name: [AD_Demo]AutomaticDiagnostics_v33_Demo
 Catalog: 1756-L85E
 Diagnostic Code: CTRL_FLT_4_34
 Message: Major Fault T04:C34 - Program Fault: A timer instruction had a negative value for its PRE or ACC.
 Device Path: DESKTOP-R9BI273\Ethernet\192.168.1.23\Backplane\0\1756-L85E\AutomaticDiagnostics_v33_Demo
 Resource: Controller
 Server Name: FactoryTalk Linx
 Product Type: Programmable Logic Controller
 Vendor: Rockwell Automation/Allen-Bradley
 Product Name: 1756-L85E LOGIX5585E
 Major Revision: 33
 Minor Revision: 4

Filter: No filter | Sorted by: Event Time(Ascending), Device Name(Ascending)

Automatic Diagnostics is enabled by default with firmware revision 33.00 or later. You can disable and enable the whole feature while online or offline from the Controller Properties dialog box. You can also disable Automatic Diagnostics for a specific device in the device's configuration.

When Automatic Diagnostics is enabled, there is little impact to device and network performance.

Notes:

Optimize an Application for Use with HMI

Rockwell Automation offers these HMI (human machine interface) platforms.

Platform	Description
PanelView™ 5000 terminal	Dedicated, machine-level HMI running Studio 5000 View Designer® software
PanelView Plus terminal	Dedicated, machine-level HMI running FactoryTalk® View Machine Edition software
FactoryTalk® View software	Product family that consists of: <ul style="list-style-type: none">• FactoryTalk View ME (Machine Edition) software for an open, machine-level HMI; also runs on PanelView Plus terminals• FactoryTalk View Site Edition Station software for a single-workstation, supervisory-level HMI• FactoryTalk View Site Edition distributed software for a multi-server, multi-client, supervisory-level HMI"

Software products that provide plant-floor device connectivity for HMI applications include:

- RSLinx® Classic software
- FactoryTalk Linx software (formerly RSLinx Enterprise)

Linux-based Software Use of Controller Memory

The amount of memory that Linux-based software needs depends on the type of data Linux-based software reads. These equations provide a memory estimate.

IMPORTANT This topic does not apply to 5380, 5480, or 5580 controllers.

Linux-based software overhead (per connection)	_____	* 1345	=	_____	bytes (four connections by default)
Individual tags	_____	* 45	=	_____	bytes
Arrays / structures	_____	* 7	=	_____	bytes
		Total	=	_____	bytes

You can consolidate tags into an array or a structure to reduce the communication overhead and the number of connections that are used to obtain the data.

HMI Implementation Options

Method	Benefits	Considerations
Single HMI	<ul style="list-style-type: none"> All HMI/EOI support this method Limited number of controller connections No server to configure and manage Local control and monitoring 	<ul style="list-style-type: none"> Single point of failure for visualization Only one person can monitor one display at a time
Multiple, Independent HMI	<ul style="list-style-type: none"> All HMI/EOI support this method The same HMI screens can be viewed at multiple stations Multiple people can monitor different parts of system simultaneously Each HMI gets its own data No central server to configure and manage Local control and monitoring 	<ul style="list-style-type: none"> More controller connections are required Additional burden on controller to service all communication (controller resource impact) No sharing of data except through the controller Adding additional HMIs has larger increase on system
Client/Server HMI	<ul style="list-style-type: none"> The same HMI screens can be viewed at multiple stations Server provides data to multiple clients Fewer controller connections required Impact on system is smaller than with multiple HMIs Administer application at the server, not individually at the clients or multiple, independent HMIs 	<ul style="list-style-type: none"> Server is a point of failure for all HMIs, unless you implement redundancy Little communication overhead savings if each client wants different data Networking knowledge that is required

Most third-party HMIs are limited to direct communication similar to the multiple HMI method.

Guidelines for FactoryTalk View Software

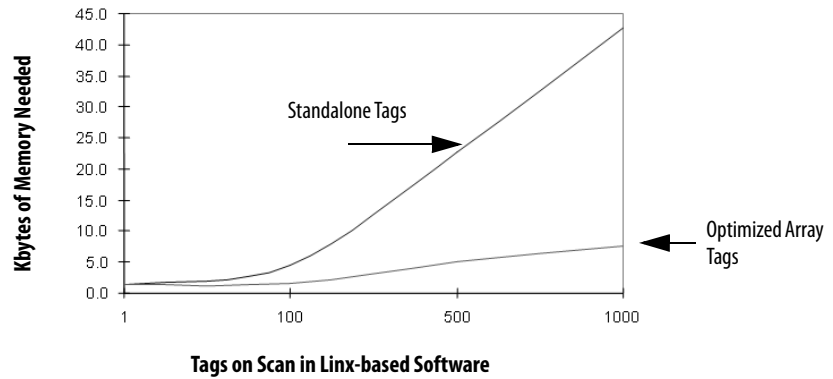
For the latest information on guidelines for FactoryTalk View Site Edition Software, see the Rockwell Automation® Knowledgebase ID 32549 titled FactoryTalk View SE Distributed System Design Considerations.

How a Data Server Communicates with the Controllers

Linux-based software acts as a data server to optimize communication to HMI applications. Linux-based software groups data items into one network packet to reduce:

- The number of messages that are sent over the network.
 - The number of messages a controller processes.
1. When Linux-based software first connects to a controller, it queries the tag database and uploads definitions for all controller-scoped tags. If there are multi-layer, user-defined structures that are controller-scoped, Linux-based software just queries the upper layer.
 2. When the HMI client requests data, Linux-based software queries the definitions for program-scoped tags and the lower layers of multi-layer user-defined structures.
 3. Linux-based software receives requests for data items from local or remote HMI/EOI clients and combines multiple requests in optimized packets. Each data item is a simple Logix tag, array, or user-defined structure. Each optimized packet can be as large as 480 bytes of data and can contain one or more data items.

4. The controller allocates unused system memory to create an optimization buffer to contain the requested data items.
 - One optimization buffer can contain as much data as can fit into one 480-byte packet (optimization is limited to 480 bytes).
 - If you use the Logix Designer application to monitor controller RAM, you can see used memory increase. (Note that this does not apply to 5380, 5480, or 5580 controllers.)
 - The controller creates an optimization buffer for each Linx-based optimization packet in the scan.



Compare RSLinx Classic and FactoryTalk Linx Software

Comparison	RSLinx Classic Software	FactoryTalk Linx Software ⁽¹⁾
Supported platforms	Microsoft® Windows operating systems. For the latest information regarding software platform support, see the Product Compatibility and Download Center: http://www.rockwellautomation.com/global/support/pcdc.page	
Data server	OPC data server Preferred data server for PLC/SLC platforms and applications that require complex network routings Maximum 10 clients per data server	FactoryTalk Live data server Preferred data server for Logix 5000™ platforms Maximum 20 clients per data server
PLC/SLC systems	Maximum 20 controllers per data server via an Ethernet network	Maximum 20 controllers per data server via an Ethernet network
Logix 5000 systems	Maximum: <ul style="list-style-type: none"> • 10 controllers per data server via an Ethernet network • 10,000 active (on-scan) tags per data server • Three RSLinx data servers per controller 	Maximum: <ul style="list-style-type: none"> • 20 controllers per data server via an Ethernet network • 20,000 active (on-scan) tags per data server • Three FactoryTalk Linx data servers per controller
User interface and event logs	Yes	<ul style="list-style-type: none"> • Available user interfaces are FactoryTalk Studio software and FactoryTalk Administration Console software • Event logs are available with FactoryTalk Diagnostics software
Benefits	<ul style="list-style-type: none"> • Supports topic switching with redundant ControlLogix® system • Supports used-defined tag optimization • RSLinx Gateway software consolidates multiple HMI requests to reduce network traffic • Works with an integrated OPC server 	<ul style="list-style-type: none"> • Automatically handles Logix tag changes • FactoryTalk Live Data software consolidates multiple HMI requests to reduce network traffic
Considerations	<ul style="list-style-type: none"> • Requires HMI to be restarted if a Logix 5000 controller is reloaded with changes to tags on scan • Default is four connections for a read and one connection for a write 	<ul style="list-style-type: none"> • Does not support topic switching with redundant ControlLogix system • Optimization is limited to array tags • FactoryTalk Gateway software provides OPC support • Default is four connections for a read and one connection for a write

(1) With version 6.00, RSLinx Enterprise is known as FactoryTalk Linx.

The Logix Designer application defaults to RSLinx Classic software. With version 31 or later, you can configure the Logix Designer application to use RSLinx Classic or FactoryTalk Linx software.

Guidelines for Linx-based Software

Guideline	Description
Use Linx-based software as the data server for multiple HMIs.	For multiple HMI stations: <ul style="list-style-type: none"> • Leverage remote OPC (RSLinx Classic software) or FactoryTalk software (FactoryTalk Linx software) for data collection. • Only the RSLinx data server is expected to have an active topic. • Do not configure or use topics on the HMI stations. • Linx-based software does not need to be on the HMI stations.
Do not use too many RSLinx stations.	The performance of tag collection decreases as the more RSLinx stations collect data from the same controller. Use an RSLinx Gateway station and have the other data collection stations use remote OPC for data collection.
Account for delay time when adding/removing scanned tags.	When switching from one HMI screen to another, it takes time to put items in the controller on scan and take items off scan. Part of this time delay is because the controller allocates system memory for the optimization buffer. To minimize this delay, when switching between HMI screens, put the items in the HMI screens on scan and leave them on scan. For example, you can create a data log to keep the items on scan. Then when switching between HMI screens, data collection continues without interruption. FactoryTalk Linx and FactoryTalk View Site Edition software account for this time delay. When HMI screens change, these applications deactivate tags rather than remove them from scan.

Guidelines to Configure Controller Tags

Guideline	Description
Use INT data types with third-party products.	Most third-party operator interface products do not support DINT (32-bit) data types. However, there are additional performance and memory-use considerations when you use INT data types. FactoryTalk View software supports native Logix 5000 data types (including BOOL, SINT, INT, DINT, and REAL), structures, and arrays.
Group related data in arrays.	Most third-party operator interface products do not support user-defined structures. Arrays also make sure that data is in contiguous memory, which optimizes data transfer between the controller and Linx-based software or other operator interfaces. Arrays of tags transfer more quickly and take up less memory than groups of individual tags.
Use Linx-based OPC services.	Use Linx-based OPC services to bundle multiple tag requests into one message to reduce communication overhead. OPC provides better optimization than DDE.

Reference Controller Data from FactoryTalk View Software

This table shows how to reference data in a FactoryTalk View tag address.

Logix 5000 Array Data Type	Description	PLC File Identifier	FactoryTalk View Tag Data Type
BOOL	Value of 0, 1, or -1	B	Digital
SINT	8-bit integer	A	Byte
INT	16-bit integer	N	Integer
DINT	32-bit integer	L	Long Integer
LINT	64-bit integer value to store date and time values	No PLC identifier	Not supported
REAL	Floating point	F	Floating Point

When addressing a Logix 5000 string tag, use the address syntax [OPC_Topic]StringTag.Data[0],SC82 to address a SINT array. The string data is stored in the SINT array .Data of the string tag, and you address the first element of this array (.Data[0]). The maximum number of characters in a STRING tag is 82. If you need more characters, then create your own user-defined structure to hold the characters.

To write data into a Logix 5000 string tag from an HMI or external source, set the L.EN field to indicate the number of characters that are in the string. The Logix Designer application and the controller use the .LEN value to determine how many characters are present.

Notes:

Develop Equipment Phases

The PhaseManager™ option of the Studio 5000 Logix Designer® application gives you a state model for your equipment. It includes the following components:

- Phase to run the state model
- Equipment phase instructions for programming the phase
- PHASE data type

Guidelines for Equipment Phases

Guideline	Description
Use a separate phase for each activity of the equipment.	Each phase is a specific activity that the equipment performs. <ul style="list-style-type: none"> • Use one phase for standalone machines. • Make sure that each phase does an independent activity. • Keep the total number of phases and programs in a project within the limit of programs for the controller. • List the equipment that goes with each phase.
Complete one state model for each phase.	Each phase runs its own set of states. A state model divides the operating cycle of the equipment into a series of states. <ul style="list-style-type: none"> • Decide which state to use for the initial state after power-up. • Start with the initial state and work through the model. • Use only the states you need; skip those states that do not apply. • Use subroutines for producing and standby states. <p>The state model of an equipment phase is similar to the S88 state model. U.S. standard ISA S88.01-1995 and its IEC equivalent IEC 61512-1-1998 is commonly referred to as S88. It is a set of models, terms, and good practices for the design and operation of manufacturing systems.</p>
Separate phase code from equipment code.	One advantage of a phase is that it lets you separate the procedures (recipes) for how to make the product from the control of the equipment that makes the product. This makes it easier to execute different procedures for different products by using the same equipment.
Separate normal execution from exceptions.	A state model makes it much easier to separate the normal execution of your equipment from any exceptions (faults, failures, off-normal conditions). <ul style="list-style-type: none"> • Use a prestate routine to watch for faults. • A prestate routine is not a phase state routine. Create a routine like you do for any program and assign it as the prestate routine for the equipment phase program. • Use a state bit to limit code to a specific state. • The Logix Designer application automatically makes a tag for each phase. The phase tag has bits that identify the state of the phase. For example, My_Phase.Running.
Use Equipment Phases in redundant systems.	PhaseManager has been tested for compatibility with ControlLogix® redundancy systems. See the ControlLogix Enhanced Redundancy System, firmware revision 16.81, Release Notes, publication 1756-RN650 , for more information.

Equipment Phase Instructions

The equipment phase instructions are available in relay ladder and structured text programming languages. You can use them in relay ladder routines, structured text routines, and SFC actions.

If you want to:	Use this instruction:
Signal a phase that the state routine is complete so go to the next state.	Phase State Complete (PSC)
Change the state or substate of a phase.	Equipment Phase Command (PCMD)
Signal a failure for a phase.	Equipment Phase Failure (PFL)
Clear the failure code of a phase.	Equipment Phase Clear Failure (PCLF)
Initiate communication with FactoryTalk® Batch software.	Equipment Phase External Request (PXRQ)
Clear the NewInputParameters bit of a phase.	Equipment Phase New Parameters (PRNP)
Create breakpoints within the logic of a phase.	Equipment Phase Paused (PPD)
Take ownership of a phase to either: <ul style="list-style-type: none"> Prevent another program or FactoryTalk Batch software from commanding a phase. Make sure another program or FactoryTalk Batch software does not already own a phase. 	Attach to Equipment Phase (PATT)
Relinquish ownership of a phase.	Detach from Equipment Phase (PDET)
Override a command.	Equipment Phase Override (POVR)

For more information, see the PhaseManager User Manual, publication [LOGIX-UM001](#).

Manage Firmware

The controllers, I/O modules, and other devices use firmware that you can update on your own. You choose the firmware revision level and decide when to update the firmware.

Guidelines to Manage Controller Firmware

Guideline	Description						
Maintain software versions and firmware revisions at the same major revision levels.	<p>At release, a specific version of software supports the features and functions in a specific revision of firmware. To use a specific revision of firmware, you must have the corresponding software version. This combination of software and firmware is considered to be compatible.</p> <p>A revision number consists of a major and minor revision number in this format xx.yyy.</p> <table> <tr> <th>Where</th><th>Is the</th></tr> <tr> <td>xx</td><td>Major revision Updated every release there is a functional change.</td></tr> <tr> <td>yyy</td><td>Minor revision Updated anytime there is a change that does not affect function or interface.</td></tr> </table>	Where	Is the	xx	Major revision Updated every release there is a functional change.	yyy	Minor revision Updated anytime there is a change that does not affect function or interface.
Where	Is the						
xx	Major revision Updated every release there is a functional change.						
yyy	Minor revision Updated anytime there is a change that does not affect function or interface.						
Use digitally signed firmware to maintain firmware integrity.	Controllers support digitally signed firmware for additional security.						
Document firmware revisions.	Include software version and firmware revision information in electrical drawings and other project documentation.						
Read the associated release notes.	Always read the release notes that accompany new software versions and firmware revisions before you install them. The release notes help you to understand what has improved and changed, and also help you determine whether you must modify your application because of the changes. In most cases, your application runs normally following an update.						
Configure modules so that the controller automatically updates firmware.	<p>Controller firmware, revision 16, includes a Firmware Supervisor feature that lets controllers automatically update devices. To use the Firmware Supervisor:</p> <ul style="list-style-type: none"> You can update Local and remote modules while in Program or Run modes, as long as their electronic keying configurations are set to Exact Match and the ControlFLASH™ Software supports the modules. Firmware kits reside on the removable media in the controller. 						
Control that users have access to change firmware revisions.	ControlFLASH software, version 8.0 and later, is integrated with FactoryTalk® Security software so you can establish update or no update privileges for users.						
Use the ControlFLASH kit manager to update only the firmware you need or have.	<p>With ControlFLASH software, version 8.0 and later, you can:</p> <ul style="list-style-type: none"> View available firmware kits before updating a device. Import and export kits to create custom kits. Delete kits as single devices or as groups by catalog number and device type. Support third-party applications to push/pull kits as needed. 						

Compare Firmware Options

Controllers ship with basic firmware that supports only updating the controller firmware to the required revision. You must update the firmware to a revision that is compatible with your version of the Studio 5000 Logix Designer® application.

ControlFLASH and ControlFLASH Plus Software	AutoFlash Function	Controller-based Firmware Supervisor
Standalone tool. Manually launch from desktop icon or program list. ControlFLASH Plus™ integrates with the Product Compatibility and Download Center and with FactoryTalk® Security software.	Integrated with the Logix Designer application. The software automatically checks the controller, motion module, and SERCOS drive firmware during a project download. If the firmware is out of date or incompatible, the software prompts you to update the firmware.	Integrated on the controller removable media and run by the controller without user intervention. Controllers automatically update modules on keying mismatch situations.
Supports controllers, communication modules, I/O modules, motion modules, and newer SERCOS drives, and many other devices. ControlFLASH Plus also supports component products.	Supports the same devices as the ControlFLASH™ Software.	Supports local and remote devices that: <ul style="list-style-type: none"> • Are in the I/O tree and configured as Exact Match. • Support firmware updates via the ControlFLASH Software. • The hardware revision supports the firmware that is stored for that Exact Match device.
Supports valid CIP™ path to the device to update, such as serial, DeviceNet®, ControlNet®, and EtherNet/IP™ connections.	Supports valid CIP path to the device to update, such as serial, DeviceNet, ControlNet, and EtherNet/IP connections.	Supports all communication paths to devices that reside in the controller I/O tree and that also support the ControlFLASH Software. The firmware must already be on removable media in the controller.

For more information, see these publications:

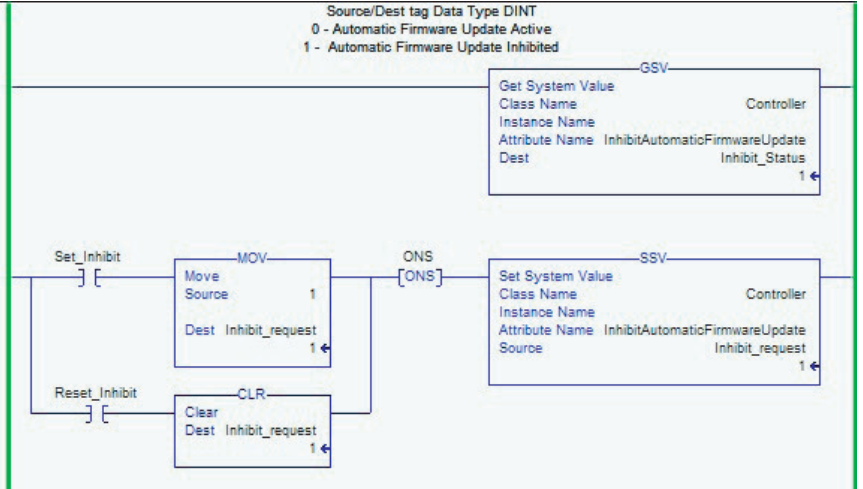
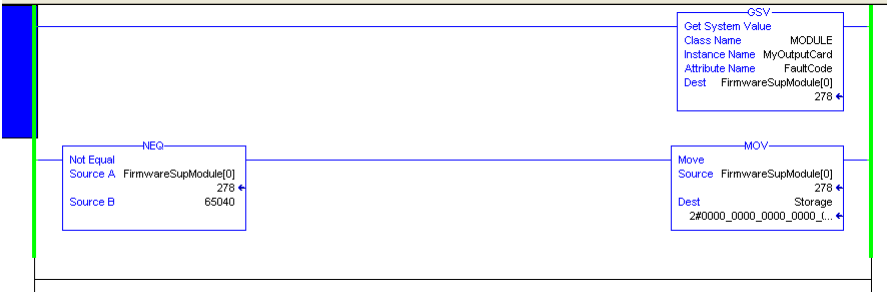
- ControlFLASH Plus Quick Start Guide, publication [CFP-QS001](#).
- ControlFLASH Firmware Upgrade Software User Manual, publication [1756-UM105](#).

Guidelines for the Firmware Supervisor

The Firmware Supervisor feature can automatically load firmware when you replace a device in the system.

- OEMs who build multiple machines a month can have the controller update all modules and devices in the system without user intervention.
- Machines with strict regulation can require specific firmware revisions for the devices to maintain certification. The Firmware Supervisor helps make sure that devices are at the correct firmware revision.
- Maintenance personnel replacing failed hardware can install the replacement device and the controller automatically updates the device with the correct firmware revision.

Guideline	Description
<p>The Firmware Supervisor can update any Rockwell Automation® device that:</p> <ul style="list-style-type: none"> • Can be placed in the I/O Configuration tree • Has electronic keying that is configured as Exact Match • Normally can be updated with ControlFLASH software 	<p>The Firmware Supervisor works on local I/O modules and distributed modules via EtherNet/IP, SERCOS, and ControlNet networks. On DeviceNet networks, the Firmware Supervisor supports local devices only, such as scanners and linking devices that reside in the I/O tree of the controller project. Because you cannot directly place a remote DeviceNet device in the I/O tree, the Firmware Supervisor does not manage remote DeviceNet devices.</p> <p>The Firmware Supervisor supports:</p> <ul style="list-style-type: none"> • Controllers that support removable media (except for redundant controllers). • The Firmware Supervisor does not manage the firmware of other standard controllers in the I/O Configuration tree. • Safety products, including GuardLogix® Safety controllers and 1791ES CompactBlock™ Guard I/O™ EtherNet/IP modules. <p>The Firmware Supervisor does not manage the firmware of POINT Guard I/O™ modules or 1791DS CompactBlock Guard I/O DeviceNet modules.</p> <p>SERCOS drives that support updates over a SERCOS network:</p> <ul style="list-style-type: none"> • 1394 drives, firmware revision 1.85 and later. • Kinetix® 6000 drives, firmware revision 1.85 and later. • Ultra™ 3000 drives, firmware revision 1.50 and later. • 8720MC drives, firmware revision 3.85 and later. <p>Non-modular, distributed I/O products that sit directly on the network without an adapter. Distributed I/O products that require an adapter, such as POINT I/O™ or FLEX™ I/O modules, are not supported. Instead, the Firmware Supervisor manages the firmware for the adapters.</p> <p>The Firmware Supervisor does not support PanelView™ Plus terminals, since the terminals do not support the ControlFLASH software.</p>
<p>For the Firmware Supervisor to manage firmware for a device, the device must have its electronic keying that is configured for Exact Match.</p>	<p>Other modules can exist in the I/O Configuration that are not configured as Exact Match, but the Firmware Supervisor does not maintain the firmware for those modules.</p> <p>To disable the Firmware Supervisor for a specific device:</p> <p>Change the electronic keying for that device to something besides Exact Match.</p> <p>Disable Firmware Supervisor from either an SSV instruction or the Nonvolatile Memory tab of the controller properties.</p>
<p>Removable media must be formatted properly.</p>	<p>If you have a Secure Digital card with 4 GB memory or more, format the card FAT32. If you have a Secure Digital card with less than 4 GB memory, format the card FAT16.</p>
<p>Make sure that the removable media is not locked.</p>	<p>The Secure Digital card has a lock feature. The card must be unlocked to write to the card.</p>

Guideline	Description
For modules that are managed by the Firmware Supervisor, each controller must store the firmware files on removable media.	<p>Enable the Firmware Supervisor, from the Nonvolatile Memory tab of the controller properties. Click Load/Store. From the Automatic Firmware Updates pull-down menu, choose Store to copy it to removable media.</p> <p>The computer that runs the Logix Designer application must have:</p> <ul style="list-style-type: none">ControlFLASH Software installed.The required firmware kits in the ControlFLASH default directory for the modules the Firmware Supervisor is to maintain. The Logix Designer application moves firmware kits from your computer to the removable media in the controller for the Firmware Supervisor to use.Controller firmware and application logic are managed outside of Firmware Supervisor on the Nonvolatile Memory tab. Firmware Supervisor adds to the ability to store controller firmware and logic on the removable media. If you disable the Firmware Supervisor, you disable the Firmware Supervisor updates and not the controller firmware updates that still occur when the controller image is reloaded.
Enable or disable the automatic firmware updates by using GSV and SSV instructions.	
You can monitor the status of automatic firmware updates.	<p>Monitor the status of automatic firmware updates on the Nonvolatile Memory tab on the controller properties. To monitor the status of automatic firmware updates for a specific module, use GSV instructions. This example shows that the Firmware Supervisor encountered the wrong hardware revision for 1756-OB16D module.</p> 

Access Firmware

The Logix Designer application ships with firmware update kits. Firmware revisions are also available on the Rockwell Automation website.

- Go to <https://www.rockwellautomation.com/global/support/download-center/overview.page?>
- Under Drivers and Firmware, click Find Drivers and Firmware.

A

access
 firmware 132
 module object 36
access the module object 36
add-on instruction
 guidelines 53
 postscan logic 41
 prescan 40
addresses
 serial bit 81
alias tags
 creating 84
applications
 HMI 121
array
 guidelines 77
 index
 guidelines 78
 indirect addresses 77
 tag storage 76
atomic data types 75
automatic diagnostics 119

B

base tag
 guidelines 83
bit tags 80
block-transfer messages
 guidelines 113
buffer
 message storage 110
 routine 39

C

cache
 messages 110
code reuse
 guidelines 46
communication
 Linux-based software data packets 122
 module connections 24
 MSG instruction 109
comparison
 import/export, add-on instructions 56
 program parameters, add-on instructions 58
 programming languages 38
 scheduled and unscheduled ControlNet 106
 subroutines, add-on instructions 55
compound data types 75
configuration
 Logix-based alarms 118
 tags 83
connection
 communication module 24
considerations
 periodic, event tasks 36
 task 34

consumed tag
 event task 74
continuous
 task 33
 lowest priority 32
controller
 dual-core 23
 Linux-based software
 software memory 121
 resources 23
 -scoped tags 122
 tag guidelines 125
ControlNet network
 guidelines 104
 scheduled and unscheduled comparison 106
 topology 103
creating
 alias tags 84

D

data
 scope guidelines 85
 type guidelines 75
DeviceNet network
 guidelines 107
 topology 106
diagnostics 119
dual-core
 controller 23

E

equipment phases 127
 guidelines 127
 instructions 128
EtherNet/IP network
 guidelines 103
 topology 102
event
 task 33
 configuration 35
 considerations 36
 consumed tag 74
 guidelines 36
execution
 project 34
 timer 42

F

FactoryTalk
 software guidelines 122
FactoryTalk Linux software 124
firmware
 access 132
 management 129
 options 130
 supervisor guidelines 131

G**guidelines**

- block-transfer messages 113
- controller firmware 129
- controller tags 125
- ControlNet network 104
- DeviceNet network 107
- equipment phases 127
- EtherNet/IP network 103
- FactoryTalk View software 122
- firmware supervisor 131
- Linux-based software 124
- messages 112

H**HMI**

- optimization 121

I**indexed routine 39****inline duplication 38****instructions**

- equipment phases 128

L**Linux-based software**

- controller memory estimate 121
- guidelines 124
- network data packet 122

logic

- routine application code 37

Logix-based

- alarm
- configuration 118

M**manage**

- firmware updates 129
- system overhead 28

map tags 113**memory**

- Linux-based software estimation 121

message

- block-transfer guidelines 113
- cache 110
- guidelines 112
- storage buffer 110

module object 36

- path attribute 37

MSG

- communication 109

N**network**

- ControlNet guidelines 104
- ControlNet topology 103
- DeviceNet guidelines 107
- DeviceNet topology 106
- EtherNet/IP guidelines 103
- EtherNet/IP topology 102
- guidelines 101
- services 101
- unscheduled and scheduled ControlNet 106
- unscheduled ControlNet guidelines 105

O**online addition of module 99****P****packet**

- Linux-based software data 122

path attribute 37**periodic**

- task 33
- configuration 35
- considerations 36

phases

- equipment 127
- PhaseManager option 127

postscan

- add-on instruction 41
- SFC logic 41

pre-defined tasks

- process controllers 33

prescan

- add-on instruction 40

priority level

- task 32

process controllers 33**produced and consumed**

- RPI 73
- tag guidelines 72
- tags 71

program

- considerations 31
- languages comparison 38
- methods 38
- scoped tags 122

project

- execution 34

R**routine**

- considerations 31
- programming logic 37

RPI

- produced and consumed tags 73

RSLinux Classic software 124**RSLinux. See also Linux-based Software**

S

serial bit addresses 81

services

network 101

SFC

logic postscan 41

online editing 43

storage

message buffer 110

string data types

guidelines 82

system overhead

manage timeslice 28

timeslice 26

T

table

mapping 113

tag

configuration 83

controller-scoped 122

descriptions 86

maps 113

name guidelines 85

produced and consumed 71

program-scoped 122

task

considerations 31, 34

continuous, periodic, event 33

priority level 32

types 32

timer execution 42

timeslice

manage system overhead 28

system overhead 26

topology

ControlNet network 103

DeviceNet network 106

EtherNet/IP network 102

U

UDT

guidelines 79

unscheduled ControlNet

network guidelines 105

updating

firmware 129

Notes:

Notes:

Rockwell Automation Support

Use these resources to access support information.

Technical Support Center	Find help with how-to videos, FAQs, chat, user forums, and product notification updates.	rok.auto/support
Knowledgebase	Access Knowledgebase articles.	rok.auto/knowledgebase
Local Technical Support Phone Numbers	Locate the telephone number for your country.	rok.auto/phonesupport
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, CompactLogix, Compact 5000, CompactBlock, ControlFLASH, ControlFLASH Plus, ControlLogix, ControlLogix-XT, Data Highway Plus, DH+, expanding human possibility, FactoryTalk, FLEX, GuardLogix, Guard I/O, Logix 5000, MicroLogix, PanelBuilder, PanelView, PhaseManager, POINT Guard I/O, POINT I/O, LC-2, PLC-3, PLC-5, Rockwell Automation, RSLinx, RSLogix 5000, RSNetWorx, RSView, SLC, Studio 5000 Logix Designer, and SynchLink are trademarks of Rockwell Automation, Inc.

CIP, CIP Sync, ControlNet, DeviceNet, EtherNet/IP are trademarks of ODVA, Inc.

Microsoft is a trademark of Microsoft Corporation.

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) 2 663 0600, Fax: (32) 2 663 0640

ASIA PACIFIC: Rockwell Automation, Level 14, Core F, Cyberport 3, 100 Cyberport Road, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846

Publication 1756-RM094K-EN-P - October 2020

Supersedes Publication 1756-RM094J-EN-P - September 2019

Copyright © 2020 Rockwell Automation, Inc. All rights reserved. Printed in the U.S.A.