

Pointe
→
Controller

**Pointe Controller
User Guide**

**Nematron Corporation
February 2003**

Nematron, OpenControl, Pointe Controller, PointeControl, Optimization, and OptiLogic are trademarks of Nematron Corporation. All other trademarks are the property of their respective holders.

Information in this document is subject to change without notice and does not represent a commitment on the part of Nematron Corporation. No part of this manual may be reproduced or transmitted in any form or by any means for any purpose without the express written permission of Nematron Corporation.

Document 5.60.00 (2/3/2003 draft)

NEMATRON CORPORATION

5840 Interface Drive
Ann Arbor, MI 48103-9515
United States

Tel: 734-214-2000
Fax: 734-994-8074

NEMATRON, LTD.

1 The Briars
Waterberry Drive
Waterlooville, Hampshire
PO7 7YH
United Kingdom

Tel: +44 23 9226 8080
Fax: +44 23 9226 8081

OPTIMATION, INC.

2800 Bob Wallace Avenue, Suite L3
Huntsville, AL 35805-4157
United States

Tel: 256-883-3050
Fax: 256-883-3070

www.nematron.com

Chapter 1: Introducing the Pointe Controller..... 13

- 1.1 The Node Controller Concept 14
- 1.2 Typical Applications 17
- 1.3 Architecture Options 18
 - 1.3.1 Stand-alone Operation 18
 - 1.3.2 Master Controller 18
 - 1.3.3 Network Node 19
- 1.4 Available Parts and Accessories 20

Chapter 2: Initial Setup..... 22

- 2.1 Getting to Know the Pointe Controller Base..... 23
- 2.2 Supplying Power to the Pointe Controller 24
- 2.3 Installing the PointeControl Software..... 25
- 2.4 Configuring the Controller’s Network Settings 26
- 2.5 Installing I/O Modules in the Controller..... 30
- 2.6 Connecting the Controller to Your PC..... 31

Chapter 3: Quickstart Project..... 32

- 3.1 Starting a New Project 33
- 3.2 Defining Input, Output, and Memory Tags 34
 - 3.2.1 Defining input bits 34
 - 3.2.2 Defining output bits 36
 - 3.2.3 Defining memory tags 37
- 3.3 Associating Tags with I/O Points..... 38
- 3.4 Creating Your First Flow Chart 42
- 3.5 Inserting a Second Decision Block..... 47
- 3.6 Assigning Outputs 48
- 3.7 Adding a Time Delay 53
- 3.8 Checking the Chart Integrity 55
- 3.9 Building the Project Runtime 56
- 3.10 Downloading and Running Your Program 59
- 3.11 Monitoring Your Program While It Runs..... 63

3.12	Setting Breakpoints.....	66
Chapter 4: System Design and Installation.....		70
4.1	Safety Guidelines.....	71
4.2	Getting to Know the Pointe Controller Base	73
4.2.1	PTC-5800 Pointe Controller Technical Description	74
4.3	Supplying Power to the Pointe Controller.....	76
4.4	Installing the PointeControl Software	77
4.5	Addressing the Pointe Controller.....	78
4.5.1	IP Address.....	78
4.5.2	Modbus Address.....	82
4.6	Overview of OptiLogic I/O	83
4.6.1	Digital Inputs.....	84
4.6.2	Digital Outputs	87
4.6.3	Analog Inputs.....	89
4.7	Determining Your I/O Needs.....	93
4.7.1	Available I/O Modules.....	93
4.7.2	Available Operator Panels.....	94
4.7.3	Calculating Your Power Budget	94
4.8	Installing I/O Modules in the Controller	96
4.8.1	Slot Numbering.....	96
4.8.2	Installing Modules.....	97
4.9	Mounting Guidelines	98
4.9.1	Mounting the Base	98
4.9.2	Mounting the Controller Base to an Operator Panel	99
4.10	Connecting the Pointe Controller to Your Network.....	101
4.10.1	Point-to-Point Connection.....	101
4.10.2	Single Hub and Switched Connections	102
4.11	Ethernet Connection Guide	103
4.11.1	UTP Cable Characteristics.....	103
4.11.2	Cable Connectors	103
4.11.3	10Base-T Connections	104
4.11.4	Straight-through Patch Cable.....	104
4.11.5	Crossover Patch Cable	104

Chapter 5: Developing Controller Programs	105
5.1 Basic Concepts in PointeControl.....	106
5.1.1 Multiple Programming Languages.....	106
5.1.2 Memory Allocation and Access.....	106
5.1.3 The Scan Cycle	107
5.2 The Visual Framework Editor (VFE).....	108
5.2.1 The Framework Editor toolbar	109
5.2.2 The Project Workspace pane	109
5.2.3 The Object Editor pane.....	110
5.2.4 The Messages pane	110
5.3 Managing PointeControl Projects.....	111
5.3.1 Creating and opening projects.....	111
5.3.2 Importing and exporting projects.....	111
5.3.3 Documenting your project.....	112
5.4 Defining Variables in Logic Memory.....	113
5.4.1 Java reserved words.....	113
5.4.2 Defining Input, Memory, and Output tags.....	114
5.4.3 Defining strings in Logic Memory.....	116
5.4.4 Defining timers in Logic Memory	117
5.4.5 Importing and exporting databases.....	118
5.5 Associating Tags with I/O points	121
5.5.1 Specifying your installed hardware.....	121
5.5.2 Configuring I/O modules	127
5.5.3 Configuring operator panels.....	128
5.5.4 Configuring additional OptiLogic RTUs	128
5.6 Building and Editing Flow Charts	129
5.6.1 Creating a new Flow Chart.....	131
5.6.2 Navigating the Flow Chart editor	132
5.6.3 Placing and configuring Flow Chart blocks.....	135
5.6.4 Building logical expressions.....	136
5.6.5 Moving, resizing, and deleting blocks in a Flow Chart.....	143
5.6.6 Adding comments to a Flow Chart.....	145
5.6.7 Logging changes in a Flow Chart.....	146
5.6.8 Making a Flow Chart a reusable Subchart.....	147

5.7	Types of Flow Chart Blocks.....	151
5.7.1	Process Block.....	151
5.7.2	Terminator Block.....	154
5.7.3	Condition (If/Then/Else) Block.....	156
	Repeat/Until Loop Block.....	158
5.7.5	While/Do Loop Block.....	159
5.7.6	Subchart Block	161
5.8	Building and Editing Ladder Diagrams.....	163
5.8.1	Creating a new Ladder Diagram.....	163
5.8.2	Navigating the Ladder Diagram editor	164
5.8.3	Adding new rungs and branches to a Ladder Diagram	166
5.8.4	Placing and configuring a Ladder Diagram block	167
5.8.5	Moving, copying, and deleting elements in a Ladder Diagram.....	168
5.8.6	Adding comments to a Ladder Diagram	169
5.8.7	Making a Ladder Diagram a reusable Sub-Ladder	170
5.9	Types of Ladder Diagram Blocks.....	171
5.9.1	Relays and Coils	171
5.9.2	Timer and Counter Blocks.....	172
5.9.3	Math Blocks	172
5.9.4	Comparison Blocks.....	173
5.9.5	Logical and Bit Shift Blocks	174
5.9.6	Selection Blocks	174
5.9.7	String Blocks	175
5.9.8	Flow Control Blocks.....	176
5.9.9	Miscellaneous Blocks.....	176
5.10	Other Framework Editor Tools.....	177
5.10.1	Finding and replacing text	177
5.10.2	Zooming in and out on a chart	177
5.10.3	Viewing tag cross references	178
5.11	Compiling your PointeControl project.....	179
5.11.1	Configuring your project's Chart List	179
5.11.2	Setting your project's scan interval.....	180
5.11.3	Checking your project's chart integrity	181
5.11.4	Building your project's runtime module	181
5.11.5	Activating the PointeControl Monitor	182

Chapter 6: Downloading to the Controller..... 183

- 6.1 Launching the PointeControl Monitor 184
- 6.2 Selecting and Attaching a Controller 185
 - 6.2.1 Detaching from a controller 186
- 6.3 Downloading a Project to the Controller 187
 - 6.3.1 Unloading a project 187
- 6.4 Starting and Stopping a Loaded Project 188
 - 6.4.1 Stopping a project 188
 - 6.4.2 Restarting a stopped project 188
 - 6.4.3 Enabling and disabling I/O 189
- 6.5 Assigning a Password to the Controller 190
 - 6.5.1 Overriding a password 191
- 6.6 Saving a Project from the Controller 192

Chapter 7: Monitoring and Debugging 193

- 7.1 Monitoring a Running Project 194
 - 7.1.1 The Charts Tab 194
 - 7.1.2 The Browser Tab 195
 - 7.1.3 The Console Tab 198
 - 7.1.4 The Controller Log Tab 201
- 7.2 Checking System Performance 202
 - 7.2.1 Scanning 202
 - 7.2.2 Loading 203
 - 7.2.3 Errors 204
- 7.3 Viewing and Debugging Charts 205
 - 7.3.1 The Debugger Window 206
 - 7.3.2 Zooming In and Out on a Chart 207
 - 7.3.3 Viewing Subcharts within a Chart 207
 - 7.3.4 Enabling Logic Flow in a Chart 208
 - 7.3.5 Enabling Debug Trace in a Chart 208
 - 7.3.6 Inserting Breakpoints in a Chart 210
 - 7.3.7 Continuing Execution after a Breakpoint 211
 - 7.3.8 Forcing New Tag Values 212
 - 7.3.9 Additional Tools for Flow Charts Only 215

Chapter 8: Networked Operations	216
8.1 Networking via OptiLogic Remote I/O	217
8.2 Networking via Modbus Data Mapping	218
8.2.1 Modbus Address	219
8.2.2 Types of Modbus data	220
8.2.3 Enabling the Modbus driver	221
8.2.4 Mapping variables to Modbus addresses	222
Chapter 9: Troubleshooting.....	227
9.1 LED Boot Indicators	228
9.2 Hardware Reset	230
Appendix A: OptiLogic Technical Specifications	231
A.1 OL2104 Relay Output Module	232
A.1.1 OL2104 Configuration Options	233
A.2 OL2108 Relay Output Module	235
A.2.1 OL2108 Configuration Options	236
A.3 OL2109 DC Sinking Output Module	239
A.3.1 OL2109 Configuration Options	240
A.4 OL2111 AC Solid State Relay Module	243
A.4.1 OL2111 Configuration Options	244
A.5 OL2201 Digital Input Simulator Module	247
A.5.1 OL2201 Configuration Options	248
A.6 OL2205 AC/DC Input Module.....	249
A.6.1 OL2205 Configuration Options	250
A.7 OL2208 DC Digital Input Module	252
A.7.1 OL2208 Configuration Options	254
A.8 OL2211 AC Digital Input Module	255
A.8.1 OL2211 Configuration Options	256
A.9 OL2252 Dual Pulse Counter.....	258
A.9.1 OL2252 Configuration Options	261
A.10 OL2258 High Speed Pulse Counter.....	264
A.10.1 OL2258 Configuration Options	268
A.11 OL2304 Analog Voltage Output Module	273

- A.11.1 OL2304 Configuration Options..... 275
- A.12 OL2408 Analog Voltage Input 276
 - A.12.1 OL2408 Configuration Options..... 277
- A.13 OL2418 Analog Current Input..... 279
 - A.13.1 OL2418 Configuration Options..... 280
- A.14 OL2602 Dual Serial Port Module 282
 - A.14.1 OL2602 Configuration Options..... 283
- A.15 OL3406 Pushbutton/Indicator Panel..... 284
 - A.15.1 OL3406 Configuration Options..... 284
- A.16 OL3420 Operator Terminal 288
 - A.16.1 OL3420 Configuration Options..... 288
- A.17 OL3440 Display Panel 291
 - A.17.1 OL3440 Configuration Options..... 291
- A.18 OL3850 Keypad Terminal 292
 - A.18.1 OL3850 Configuration Options..... 292

Appendix B: Flow Chart Command Reference..... 297

- B.1 General Commands..... 298
 - B.1.1 Turn On and Turn Off 298
 - B.1.2 Assign..... 298
 - B.1.3 Increment and Decrement 299
 - B.1.4 Clear 300
 - B.1.5 Enable and Disable..... 300
 - B.1.6 Get Tag Name..... 301
 - B.1.7 Wait..... 302
- B.2 Timer Commands..... 303
 - B.2.1 Timer Start and Timer Stop..... 303
 - B.2.2 Timer Reset..... 303
 - B.2.3 Timer Preset 304
- B.3 String Commands 306
 - B.3.1 String Copy..... 306
 - B.3.2 String Concat 307
 - B.3.3 String Left and String Right..... 308
 - B.3.4 String Mid 309

B.3.5	String Insert	310
B.3.6	String Delete.....	311
B.3.7	String Replace	313
B.3.8	String Format Integer	314
B.4	Diagnostics Commands.....	316
B.4.1	Diag Get Tag Status.....	317
B.4.2	Diag Set Tag Status.....	318
B.4.3	Diag Clear Tag Status.....	319
B.5	Serial Commands.....	321
B.5.1	Serial Configure Port.....	321
B.5.2	Serial Enable Port and Serial Disable Port.....	323
B.5.3	Serial Read Byte.....	324
B.5.4	Serial Write Byte.....	325
B.5.5	Serial Read MultiBytes.....	326
B.5.6	Serial Write MultiBytes.....	327
B.5.7	Serial Get Comm Errors	328
B.6	Date/Time Commands.....	330
B.6.1	Date/Time Get	330
B.6.2	Date/Time Format	331
B.6.3	Get Elapsed Time.....	332
B.7	Operator Panel Commands	333
B.7.1	Keypad Data Entry.....	333
B.7.2	Arrow Adjust Data Entry.....	334
B.7.3	Button On and Button Off	336
Appendix C: Ladder Diagram Block Reference		337
C.1	Relays and Coils	338
C.1.1	Normally Open Contact (XIC)	338
C.1.2	Normally Closed Contact (XIO).....	338
C.1.3	Rising Edge Relay (LEC)	339
C.1.4	Falling Edge Relay (TEC).....	339
C.1.5	Output Coil (OC).....	340
C.1.6	Negated Output Coil (NEGOC).....	341
C.1.7	Latched Coil (LOC).....	341
C.1.8	Unlatched Coil (UOC).....	342
C.1.9	Rising Edge Coil (LEOC)	343

C.1.10	Falling Edge Coil (TEOC)	343
C.1.11	Falling Edge Detector (F_TRIG)	344
C.1.12	Rising Edge Detector (R_TRIG)	345
C.1.13	Set-Dominant Bistable (SR)	346
C.1.14	Reset-Dominant Bistable (RS)	348
C.2	Timer and Counter Blocks	350
C.2.1	Timer, Pulse (TP)	350
C.2.2	Timer, ON Delay (TON)	351
C.2.3	Timer, OFF Delay (TOF)	352
C.2.4	Counter, Up (CTU)	353
C.2.5	Counter, Down (CTD)	355
C.2.6	Counter, Up/Down (CTUD)	356
C.3	Math Blocks	359
C.3.1	Add (ADD)	359
C.3.2	Subtract (SUB)	360
C.3.3	Divide (DIV)	362
C.3.4	Multiply (MUL)	363
C.3.5	Square Root (SQRT)	364
C.3.6	Modulus (MOD)	366
C.3.7	Sine (SIN)	367
C.3.8	Cosine (COS)	368
C.3.9	Tangent (TAN)	369
C.3.10	Arc Sine (ASIN)	370
C.3.11	Arc Cosine (ACOS)	371
C.3.12	Arc Tangent (ATAN)	372
C.3.13	Absolute Value (ABS)	373
C.3.14	Logarithm (LOG)	374
C.3.15	Natural Logarithm (LN)	375
C.3.16	Exponential (EXPT)	376
C.3.17	Natural Exponential (EXP)	378
C.3.18	Expression (EXPR)	379
C.4	Comparison Blocks	381
C.4.1	Greater Than (GT)	381
C.4.2	Greater Than or Equal to (GE)	382
C.4.3	Equal to (EQ)	383
C.4.4	Not Equal to (NE)	385

C.4.5	Less than or Equal to (LE).....	386
C.4.6	Less Than (LT)	387
C.5	Logical and Bit Shift Blocks	389
C.5.1	And (AND).....	389
C.5.2	Or (OR)	391
C.5.3	Exclusive Or (XOR).....	393
C.5.4	Not (NOT)	394
C.5.5	Shift bits Left (SHL)	396
C.5.6	Shift bits Right (SHR).....	397
C.5.7	Rotate bits Left (ROL).....	399
C.5.8	Rotate bits Right (ROR)	400
C.6	Selection Blocks	402
C.6.1	Select minimum value (MIN).....	402
C.6.2	Select maximum value (MAX).....	403
C.6.3	Limit value (LIM).....	405
C.6.4	Select one of two values (SEL)	406
C.7	String Blocks	408
C.7.1	Set string (SET)	408
C.7.2	Find string length (LEN)	409
C.7.3	Extract sub-string from left (LEFT).....	410
C.7.4	Extract sub-string from right (RIGHT).....	411
C.7.5	Extract sub-string from middle (MID).....	413
C.7.6	Concatenate strings (CAT)	414
C.7.7	Compare strings (CMP)	416
C.7.8	Insert sub-string (INS).....	417
C.7.9	Delete sub-string (DEL).....	419
C.7.10	Replace sub-string (REPL).....	421
C.7.11	Find sub-string (FIND)	423
C.8	Flow Control Blocks	425
C.8.1	Call sub-ladder diagram (CALL).....	425
C.8.2	Return to main diagram (RETN).....	426
C.9	Miscellaneous Blocks	427
C.9.1	Convert to Boolean (TO_BOOL)	427
C.9.2	Convert to Integer (TO_INT).....	428
C.9.3	Convert to Float (TO_FLT)	429

C.9.4 Convert to String (TO_STRG) 430

C.9.5 Truncate (TRUNC) 432

C.9.6 Integer to Character (TO_CHR) 433

C.9.7 Character to Integer (CHR_TO) 434

C.9.8 Integer to BCD (TO_BCD) 435

C.9.9 BCD to Integer (BCD_TO) 436

C.9.10 Move (MOVE) 438

Chapter 1: Introducing the Pointe Controller

Welcome to the Pointe Controller. Nematron's Pointe Controller is an important part of the new generation of automation technology. The Pointe Controller extends automation capabilities, previously available only in high-end PC-based control systems, to low cost embedded control. It incorporates modular I/O and advanced communications features in a controller built on advanced e-control software architecture. The result is a very cost effective, easy to implement, high performance controller that can be used either stand alone or as a node in a larger e-control system.

Central to the Pointe Controller is the OpenControl software development environment. OpenControl is Nematron's industry leading PC-based control software package that has been successfully used in recent years in large, high performance manufacturing applications. It provides the means for advanced application development in Visual Flowchart Language (VFL) or ladder logic. The Pointe Controller is the result of Nematron's development effort to develop scaleable solutions. With the addition of Pointe Controller, Nematron now provides the means to address a much larger range of data acquisition and control applications.

Pointe Controller technology is both powerful and easy to use. It will enable you to quickly implement application systems ranging from simple, stand alone machine controllers to large, integrated, communications intensive systems.

1.1 The Node Controller Concept

Deployable, localized control with a high-speed, standards-compliant interface to a larger network of devices – that’s the concept of a node controller.

The Pointe Controller is part of a new generation of control devices designed to meet the needs of today’s more integrated world. It provides all of the features and functionality of traditional PLCs together with advanced features that enable it to become a node in a larger, more integrated system. That is the Node Controller Concept – deployable, localized control with a high-speed, standards-compliant interface to a larger network of devices. The design architecture enables this advanced functionality to be implemented easily and seamlessly.

All of the key features of the traditional PLC are incorporated into the design of the Pointe Controller. These key features include the following:

- Real-time local control;
- Embedded, high reliability electronic design;
- Modular I/O;
- Pluggable terminal strip connectors;
- Ladder logic programming capability; and
- Compact, DIN rail mountable package.

Pointe Controllers also incorporate many advanced features that are typically not found in traditional PLCs. Some of these advanced features include:

- Advanced visual flow chart programming capability;
- Ability to mix flowchart and ladder programming;
- Application program deployment over an Ethernet link;
- Firmware update deployment and management over an Ethernet link;
- Advanced communications for development, deployment, monitoring and coordination within a networked system;
- Remote monitoring, programming and debug capability over Ethernet;
- Powerful diagnostic tools for use during program development and system monitoring;
- Scalable solutions;
- Remote Ethernet I/O options and ability to network Pointe Controllers;
- Easy integration into larger systems and easy future expansion of systems;
- Coordinated operation within a larger system;

- Real-time, open standard data sharing from production to the front office; and
- Direct connect operator panel interfaces available.

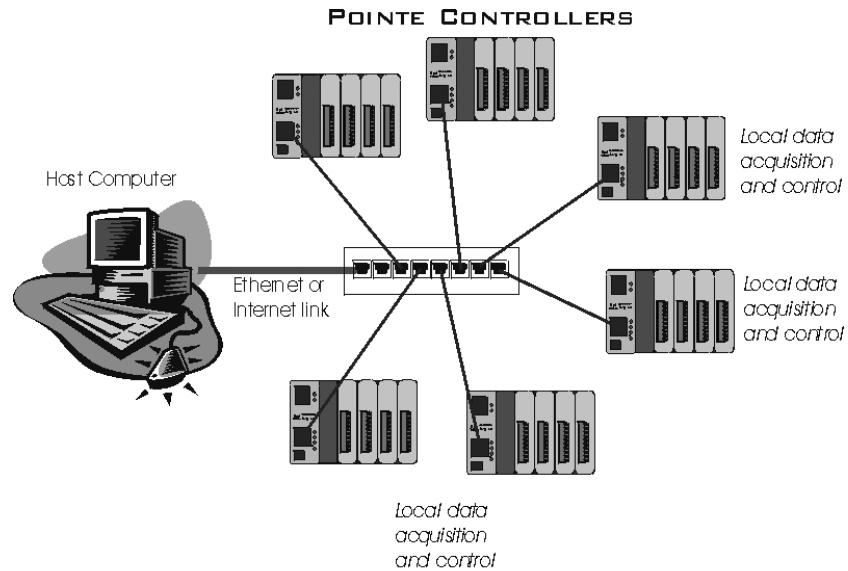
Since the mid 1970s, PLCs have been the foundation of traditional control applications. PLCs have provided the advantages of localized control, modular configuration, and rugged industrial packaging. Where traditional PLCs have been weak has been in the areas of distributed architectures, communications and more complex logic.

The 1990s brought new demands on control systems for deployability, information collection and coordination. Since traditional PLCs are weak in these types of capabilities, a new technology called “PC-based control” was launched by Nematron and adopted by a number of other automation technology companies. PC-based control has been very successful in mid-size to large systems applications with distributed I/O and integration with other software systems for e-manufacturing, MES, MRP, quality control and web access.

While “PC-based control” technology has brought the world of industrial automation a number of improvements in processing power, communications, performance and maintainability, it has had some current limitations. Those limitations have been primarily based on the economics aspects of implementing PC-based control being economically justifiable in only mid to high-end applications.

Nematron developed the Pointe Controller to capitalize on the advanced features of PC-based control while scaling the solution opportunity to medium and low-end applications. While the Pointe Controller has many applications in independent, non-networked applications, its unique design also forms the basis for control solutions for widely distributed yet integrated applications.

The figure below illustrates the power of the Pointe Controller. It shows a number of Pointe Controllers connected into a larger network. Each Pointe Controller interfaces local I/O and performs local control on a real-time dedicated basis. Each Pointe Controller can also be accessed by the larger system (shown as "Host Computer") to be polled for data and status, to coordinate applications, be reprogrammed and deployed on demand, and any other type of system-wide activity that may be desired.



The Pointe Controller's design implementation of a low cost, modular, embedded real-time control platform creates a tremendous level of opportunity and changes the way control solutions can be implemented.

1.2 Typical Applications

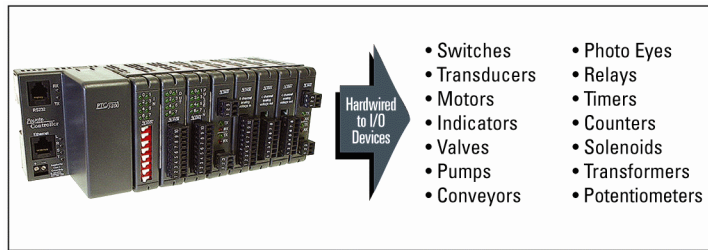
Pointe Controllers can be used for a wide variety of data acquisition and control applications. The Pointe Controller's low cost, flexible and easy to use design makes it a perfect choice for many small control applications that traditionally have used PLC type solutions. Its advanced capabilities, which emphasize deployability and network interoperability make it particularly applicable to larger, distributed control applications. Typical usage includes the following applications:

- Packaging machinery
- Semiconductor equipment
- Distributed manufacturing
- Building automation
- Data acquisition

1.3 Architecture Options

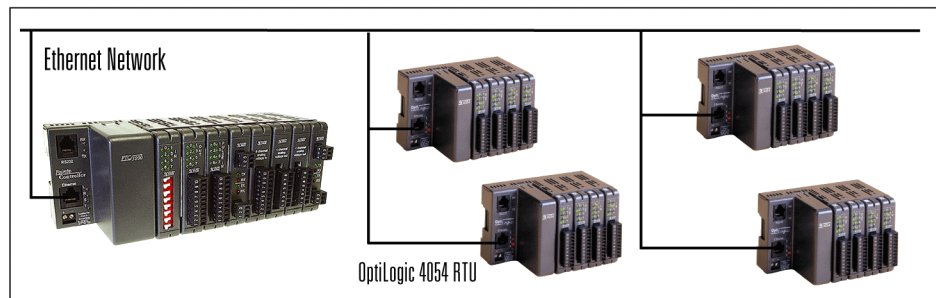
1.3.1 Stand-alone Operation

The general discussion of the Pointe Controller has emphasized the communications capability and capability to be deployed as a node on a network. In reality, the Pointe Controller is just as applicable to stand-alone applications that do not require remote communications.



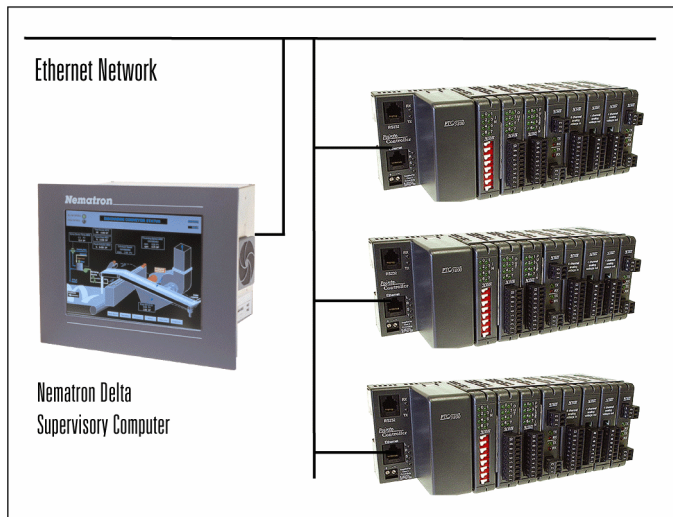
1.3.2 Master Controller

As a Master Controller for distributed control solutions, the Pointe Controller is capable of interfacing with up to four OptiLogic Remote I/O terminals via Ethernet. This reduces wiring cost of the I/O devices back to the controller, while providing high speed I/O control from the controller.



1.3.3 Network Node

As a control node in a scalable network, the Pointe Controller performs dedicated real-time local control, while maintaining communications with the designated supervisory computer. Total system deployment, configuration, project coordination, and data logging can be implemented from any authorized network workstation.



1.4 Available Parts and Accessories

The Pointe Controller system is designed to be extremely modular and flexible. As such, all Pointe Controller parts and accessories are sold separately. For availability and pricing, please call Nematron at 1-800-636-2876, or visit us on the Web at <http://www.nematron.com/Sales> and find a certified Nematron distributor near you.

A complete list of available parts and accessories follows:

PART #	DESCRIPTION
PTC-5800	Pointe Controller base unit, 8-slot NOTE: First-time users must also order at least one PointeControl software CD (part # NS-PTC) and one RS-232 download cable (part # OL-CBL-DNL). For more information, see below.
<i>Software</i>	
NS-PTC	PointeControl Development Framework and Runtime Monitor (software CD), for Pointe Controller program development and debugging. Single-seat license that can be used to program multiple Pointe Controller units. Runs on Microsoft Windows NT 4.0, Windows 2000, and Windows XP.
<i>Accessories</i>	
OL-CBL-DNL	Pointe Controller Boot Program download cable — connect's PC's RS-232 port to Pointe Controller's RS-232 port for IP address setting.
OL-CBL-X01	Ethernet CAT5 crossover cable, 6 foot, red — for direct connecting PC to Pointe Controller for program downloading.
OL-CBL-P01	Ethernet CAT5 patch cable, 6 foot, black — for connecting Pointe Controller to an Ethernet network.
CBL-PV10	Pointe Controller to Nematron PowerView™ Touchscreen HMI communication cable — supports Modbus protocol on Serial.
OL-PS1	Power Supply — 120 VAC to 24 VDC, 1 Amp output, wall pluggable.
<i>OptiLogic I/O Modules</i>	
OL0001	Blank Module — recommended to fill unused slots in base unit.
OL2104	4-point Relay Output Module, 5-30VDC or 5-132VAC, 2A/point @ 24VDC, 1A/point @ 120VAC, isolated contact outputs.
OL2108	8-point Relay Output Module, 5-30VDC or 5-132VAC, 2A/point @ 24VDC, 1A/point @ 120VAC, 4A/common, 2 commons.
OL2109	8 points, 5-40VDC sinking output. 300 mA sinking capability / point.
OL2111	8 points, solid-state relay output. 15-132VAC, 0.5A/point.
OL2201	8-point digital input simulator.
OL2205	4-point AC/DC input. 10-30V (sourcing or sinking), isolated input and return lines.

PART #	DESCRIPTION
OL2208	8-point DC input. 10-30VDC (sourcing or sinking).
OL2211	8-point AC input. 80-132VAC.
OL2252	Counter Input, 2 channels, 0-15 KHz.
OL2258	High Speed Counter. P&D, Quadrature, Up/Down Count. 0-80 KHz (0-160 KHz for Quad). Two high-speed open collector outputs.
OL2304	4-channel analog voltage output, 14-bit. 0-5v, 0-10v, +/-5v, +/-10v configurable. 12-bit resolution.
OL2408	8-channel analog voltage input, 14-bit. 0-5VDC or 0-10VDC configurable.
OL2418	8-channel analog current input, 14-bit. 4-20 mA.
OL2602	2-channel RS232 serial.
<i>OptiLogic Operator Panels and Interconnect Cables</i>	
OL3406	Pushbutton / Indicator Panel. 4 user-definable pushbuttons, 6 LED indicators. Comes with OL-CBL-RIB1.
OL3420	Terminal Panel. 2 line x 20 character alphanumeric LCD. LED backlit. 4 user-definable pushbuttons. Comes with OL-CBL-RIB1.
OL3440	Alphanumeric Display panel. 4 line x 20 character LCD. LED backlit. Comes with OL-CBL-RIB1.
OL3850	Keypad Terminal panel. 2 line x 20 character alphanumeric LCD. LED backlit. Numeric keypad. Three large LED indicator light bars. 5 user-definable pushbuttons. Comes with OL-CBL-RIB1.
OL-CBL-RIB1	OL Operator Panel interconnect cable, 1 inch (approximate). Comes with each operator panel.
OL-CBL-RIB12	OL Operator Panel interconnect cable, 12 inches (approximate).
OL-CBL-RIB36	OL Operator Panel interconnect cable, 36 inches (approximate).
<i>OptiLogic RTUs (remote I/O base units for networked systems)</i>	
OL4054	4-slot RTU base — 10BaseT Ethernet interface, 1 RS232 port, operator panel interface, DIN rail mountable, 8-30VDC power required for operation.
OL4058	8-slot RTU base — 10BaseT Ethernet interface, 1 RS232 port, operator panel interface, DIN rail mountable, 8-30VDC power required for operation.
OL4228	8-slot Modbus RTU base — 10BaseT Ethernet interface, 1 RS232 port, operator panel interface, DIN rail mountable, 8-30VDC power required for operation.

Complete technical specifications for all OptiLogic I/O modules and operator panels listed above can be found in Appendix A, starting on page 231.

Chapter 2: Initial Setup

This chapter describes how to quickly set up the Pointe Controller unit, including connecting a power supply to the Pointe Controller unit, installing the PointeControl software on your PC, setting the controller's network address, installing I/O modules in the card cage, and connecting the controller to an Ethernet network.

This setup procedure is greatly abbreviated, so that you can jump right into simple program development as described in Chapter 3, "Quickstart Project," starting on page 32. For complete system design and installation instructions, see Chapter 4, "System Design and Installation," starting on page 70.

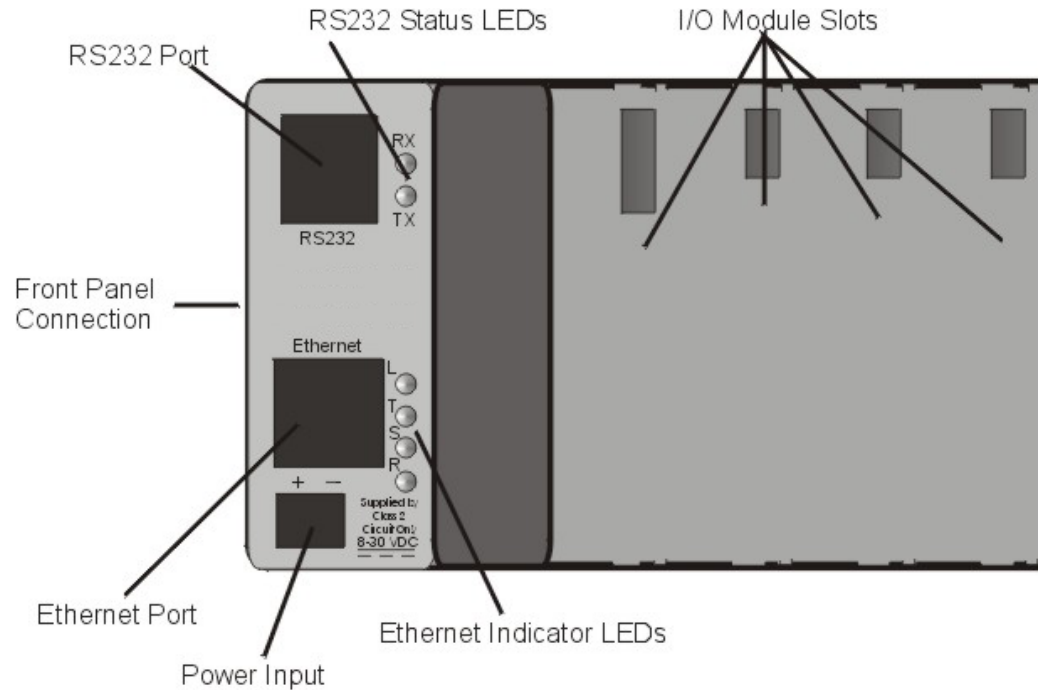
To set up a Pointe Controller unit, you must have *at least* the following items:

- A Windows-based PC that meets the PointeControl system requirements
- One PointeControl software CD (part number NS-PTC)
- One PTC-5800 base unit, 8-slot (part number PTC-5800)
- One AC power adapter and cord (part number OL-PS1)
- One RS-232 download cable, DB-9 to RJ-11 (part number OL-CBL-DNL)
- One 10BaseT crossover cable (part number OL-CBL-X01)

NOTE: All Pointe Controller parts and accessories are sold separately, as described on page 20.

2.1 Getting to Know the Pointe Controller Base

The figure below shows the layout of a Pointe Controller base:



The Pointe Controller base consists of a card cage containing the motherboard. The base unit has a built in Ethernet port, as well as an RS232 port. The Ethernet port is the interface to the larger system. The RS232 port is provided for general purpose communications (as defined by your application program). It is also designed to allow you to load future program upgrades (to incorporate the ability to interface future I/O boards and operator panels) into the base.

Both communications ports have status indicator LEDs which provide you with visible indications of each port's operation. The RS232 serial port indicates when it is transmitting (TX) and receiving (RX). The Ethernet port provides indications for good Ethernet link connection (L) and Ethernet port access by the base processor (S), as well as transmit (T) and receive (R) indicators.

Power must be provided to the unit by an external DC power supply. Any DC voltage within the range of 8-30VDC is acceptable.

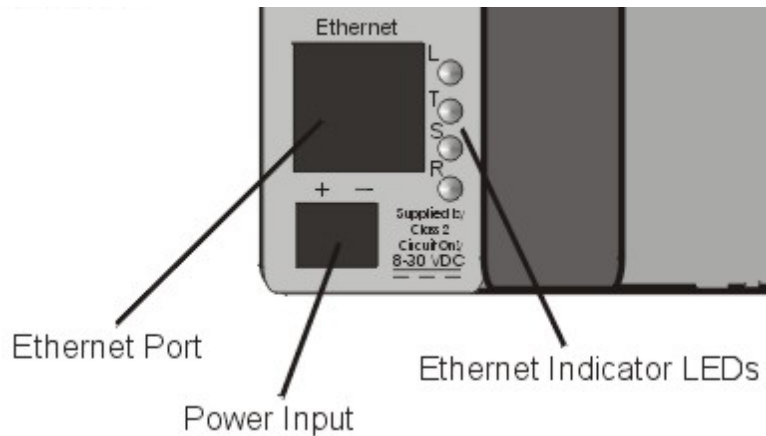
Input and output modules can be plugged into the slots in the base. Most modules can plug into any base slot (including slot 0).

NOTE: Slot 0 includes additional features used by certain 12-pin specialty modules. These modules are documented as slot 0-specific.

The OptiLogic base can snap onto any standard DIN rail, including the rail molded into the back of all OptiLogic operator panels. When attaching an OptiLogic base to an OptiLogic operator panel, the 10-pin cable connection on the side of the base is used.

2.2 Supplying Power to the Pointe Controller

The Pointe Controller unit requires a 8-30VDC, 1 Amp power supply. This can be provided either by using a wall-pluggable AC adapter (part number OL-PS1) or by connecting the unit directly to a properly rated DC grid. The connection is made at the Power Input screw terminals located at the bottom left corner of the base unit.



To connect a power supply to the controller:

1. Make sure the power is OFF – the AC adapter should be unplugged and/or the DC grid should be turned off.
2. Using a regular slotted screwdriver, loosen the Power Input screw terminals.
3. Pass the power supply wires through the opening in the bottom of the controller base unit. Insert the positive wire into the positive terminal (+) and the negative wire into the negative terminal (-).
4. Retighten the screw terminals.
5. Tug gently on the wires to verify that they are properly secured to the terminals.

The Pointe Controller unit can now be powered on.

2.3 Installing the PointeControl Software

The PointeControl software CD (part number NS-PTC) includes the control application development package and the various utilities needed to connect to and configure the Pointe Controller unit.

System requirements:

- 200 MHz or faster Pentium processor
- Operating system (any one):
 - Microsoft Windows NT 4.0 Service Pack 5 or 6
 - Microsoft Windows 2000
 - Microsoft Windows XP
- A CD-ROM drive
- A 10BaseT Ethernet card
- A DB-9 serial port

To install the PointeControl software on your PC:

1. Insert the PointeControl software CD into your CD-ROM drive.
2. Open the mounted CD (typically drive D: or E:) and double-click SETUP.EXE.
3. Follow the onscreen installation instructions. No unusual installation options are presented.
4. Restart your PC when prompted.

NOTE: As part of the PointeControl software installation, a Java Runtime Engine (JRE) is also installed on your PC. This JRE is used only by the PointeControl software and it is not included in the Windows registry. It should not conflict with any other Java tools you may have installed on your PC.

2.4 Configuring the Controller's Network Settings

Each Pointe Controller unit has two distinct addresses: an IP address, for communicating across an Ethernet network; and a Modbus address, for communicating with serial Modbus devices such as operator panels and bar code readers. In this initial setup, we will configure only the IP address.

The Pointe Controller unit comes preconfigured with a default IP address. You must reset the address so that the unit can properly communicate on your Ethernet network. This change is made via a direct serial connection between your PC and the Pointe Controller unit.

Remember that each Pointe Controller unit on your network must have its own unique IP address and node name, which is set prior to applying power to the controller. Duplicate addresses will cause system communications to fail.

To set the IP address of the Pointe Controller unit:

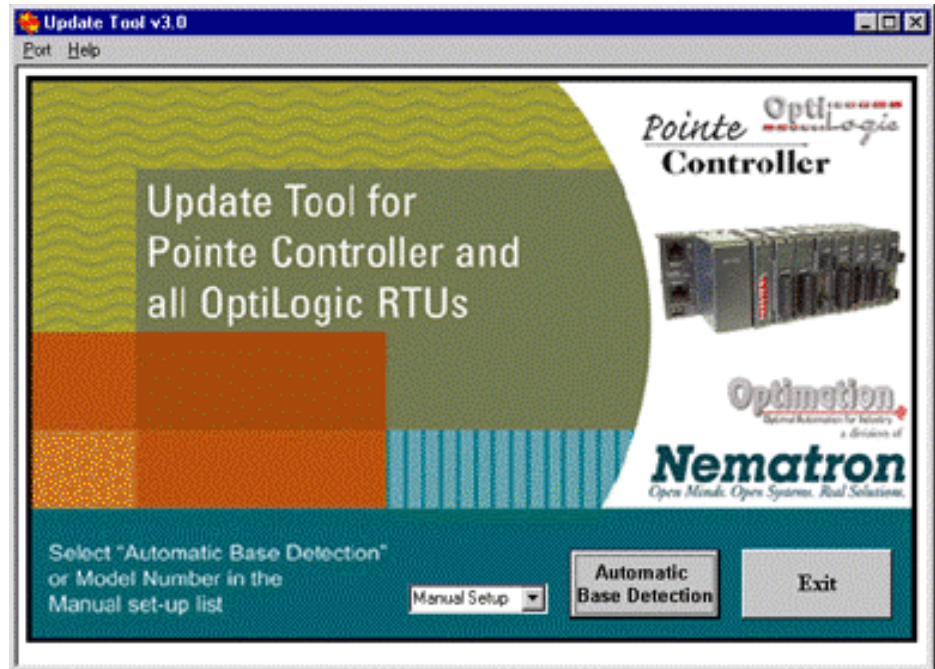
1. Establish a serial connection between your PC and the Pointe Controller unit, using the download cable (OL-CBL-DNL):
 - a. Before you connect the serial cable, make sure the Pointe Controller is powered off. The unit looks for the cable when it is first powered on.
 - b. Connect the cable's RJ-11 plug to the Pointe Controller's serial port.
 - c. Connect the cable's DB-9 plug to your PC's serial port.
2. Power on the Pointe Controller unit.

NOTE: There is no power switch on the Pointe Controller unit itself. Either the AC adapted must be plugged in or the directly connected DC grid must be turned on.

3. From the Windows **Start** menu, choose **Programs > PointeControl > Update Tool**.

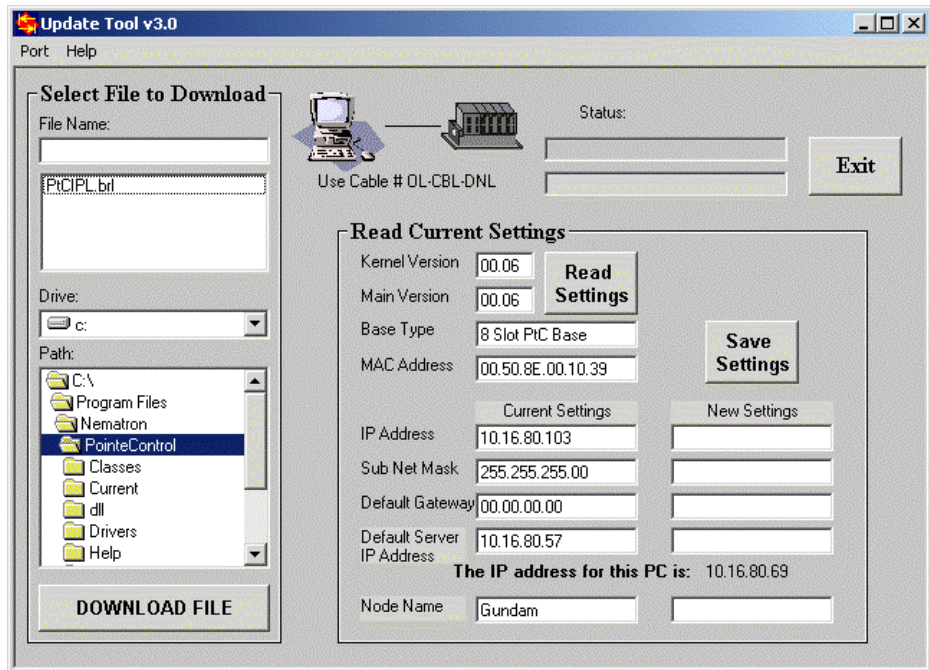
NOTE: If you are running the Update Tool for the first time, you will be asked to specify which COM (serial) port the tool should use. Enter the number (1, 2, 3, or 4) to which you connected the serial cable in Step 1 above, and then click **OK**. After that, the tool will finish launching.

The PointeControl/OptiLogic Update Tool application window appears.



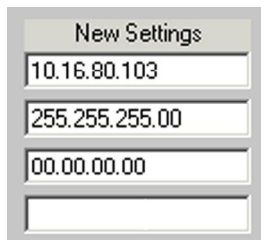
4. Click **Automatic Base Detection**. The application should immediately connect to your Pointe Controller unit. If it does not, check your serial connection and try again. If it still cannot connect, click **Manual Setup** and select **PTC5800** from the drop-down menu.

When the application successfully connects, the following window will appear:



The **Read Current Settings** pane displays the current address settings on the Pointe Controller unit. If you are addressing the unit for the first time, the factory default settings are displayed.

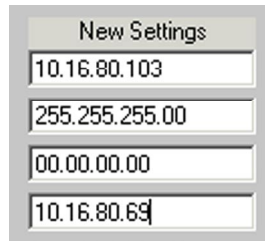
5. Under **New Settings**, enter the new IP address and subnet mask for the Pointe Controller unit. For example, an IP Address of "10.16.80.103" and a Sub Net Mask of "255.255.255.00".



The Pointe Controller unit should receive an address on the same subnet as your PC. If you do not know what values to enter, contact your system administrator.

NOTE: The Pointe Controller unit does not communicate directly with any network gateway or router. Instead, it broadcasts to all machines on its subnet. Therefore, you should enter "00.00.00.00" in the Default Gateway field.

6. For the **Default Server IP Address**, enter the IP address of the PC with which you are connecting to the Pointe Controller unit. For example, "10.16.80.69".



New Settings
10.16.80.103
255.255.255.00
00.00.00.00
10.16.80.69

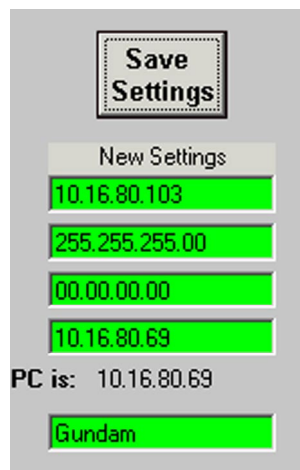
The secondary server is the PC to which the Pointe Controller unit will attempt to connect when it first powers on.

7. For the **Node Name** pane, enter the name by which the Pointe Controller unit will identify itself to PointeControl Monitor. For example, "Gundam."

NOTE: If you do not want or need to change the Node Name, you can skip this step and leave the factory default setting.

For more information on PointeControl Monitor, see Chapter 6, "Downloading to the Controller," and Chapter 7, "Monitoring and Debugging."

8. Click the **Save Settings** button to save your settings to the Pointe Controller unit. When the settings are saved, the fields will turn green.



New Settings
10.16.80.103
255.255.255.00
00.00.00.00
10.16.80.69

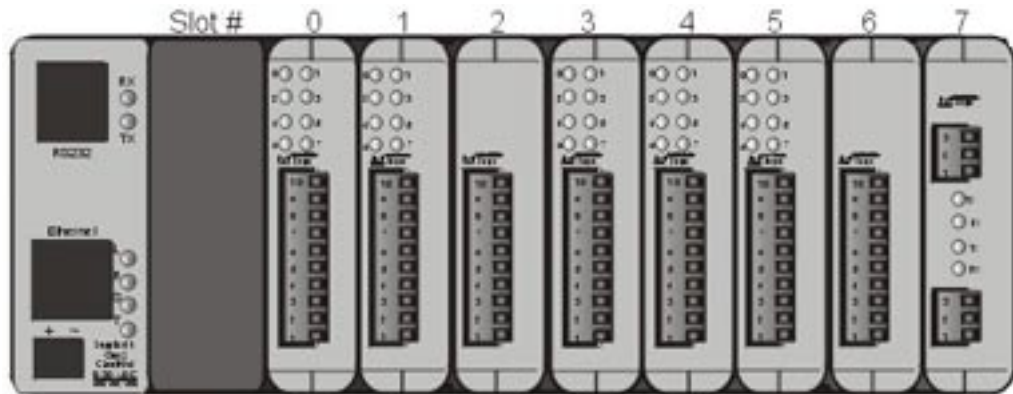
PC is: 10.16.80.69

Gundam

9. Click the **Read Current Settings** button to verify that the new settings were saved correctly. The current settings should now match the IP address, subnet mask, secondary server, and node name that you entered.
10. Exit the Update Tool application by clicking the **Exit** button.
11. Power off the Pointe Controller unit.
12. Disconnect the RS-232 serial cable.

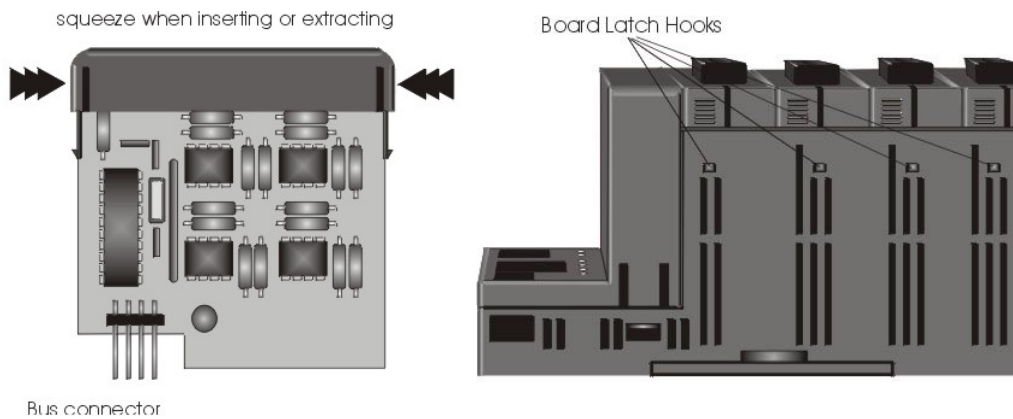
2.5 Installing I/O Modules in the Controller

Each module occupies one slot in the controller base. Each slot position is numbered as shown below. The slot number will provide a reference to your application program for selecting the appropriate module for each particular operation.



Slot numbering is simply left to right, starting with slot number 0.

Each slot has card guides along each side and a connector on the motherboard. To install an I/O module, place the module's circuitry board in the top and bottom card guides. (Note that the board will not be tightly retained until it is approximately 3/4 inch into the card guide.)

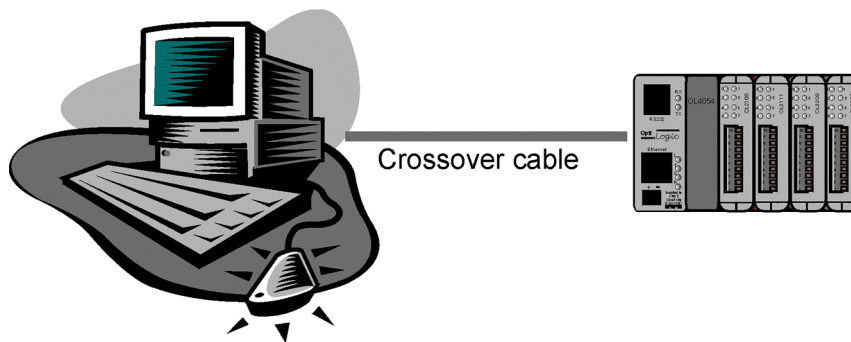


As you push the module into its mating connector, squeeze the ends together. This will allow the board latches to travel inside the card cage. When you have pushed the board into its mating connector and released, the latches should hook the card cage and keep the module in place.

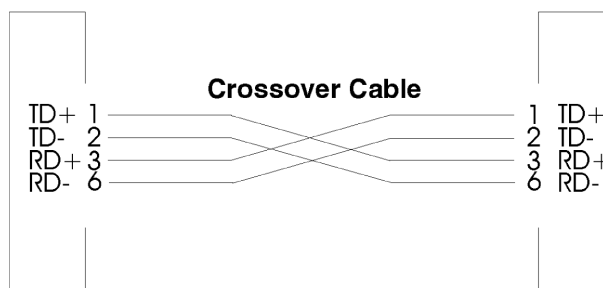
2.6 Connecting the Controller to Your PC

An Ethernet network connection is used to download finished control applications from your PC to the Pointe Controller unit. It is also used to monitor the unit's runtime performance and to share Modbus TCP data between different control devices. An RJ-45 Ethernet port is located at the left side of the Pointe Controller unit.

The simplest system is a point-to-point connection. Point to point connections, as illustrated below, require only a *crossover* type patch cable.



An Ethernet crossover cable, shown below, connects the transmitter on one side, with the receiver on the other. This is a category 5 type UTP crossover patch cable. Cable length is limited to less than 100 meters.



You can now power on the Pointe Controller unit and proceed to Chapter 3, "Quickstart Project."

Chapter 3: Quickstart Project

The first project we will implement with the Pointe Controller is a very simple one that involves setting up an automated timer and then adding a few different types of indicators that can be used to display the progress of the timer. To implement this project, you will need the following.

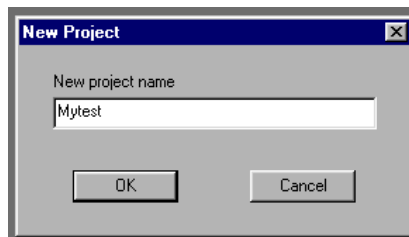
- A Pointe Controller base unit, configured as described in “Initial Setup” above
- An OL2201 Digital Input Simulator module, installed in the first I/O slot
- An OL2109 Digital Output module, installed in the second I/O slot

The program that we are going to develop will monitor the first toggle switch on the OL2201 input module. If it is on, then the program will start the timer and display its progress using the LEDs of the OL2109 output module and the four-line LCD of the OL3440 display panel.

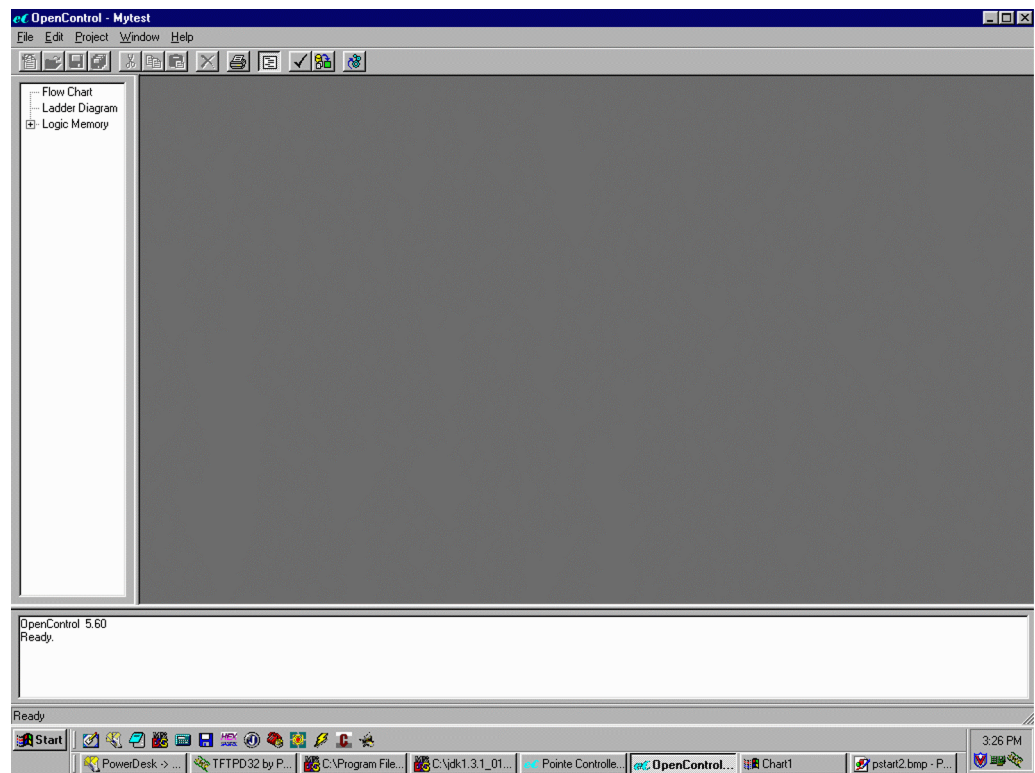
3.1 Starting a New Project

Begin by launching the PointeControl Framework application and creating a new project:

1. From the Windows **Start** menu, choose **Programs > PointeControl > Framework**. The PointeControl Framework window will appear.
2. From the **File** menu, choose **New Project**.
3. In the New Project dialog box, enter the name of your project (e.g. "Mytest") and click **OK**.



Once you have done this, your new project will be opened. The project name will be shown on the title bar on the top of the PC screen and a project tree, containing Flow Chart, Ladder Diagram and Logic Memory definitions (all undefined at this time) will show up in the left hand pane of the display.



3.2 Defining Input, Output, and Memory Tags

The next thing that we'll do is define our inputs, outputs and memory tags that we'll need for this project. In general, you can do I/O and data memory definitions piecemeal, throughout the project development process – whenever you find you need another variable or I/O point.

In this project, we will define all of the I/O points — eight input switches and eight output LEDs — even though we'll be using only one input switch. We will also attach four string variables to the four lines of the display panel.

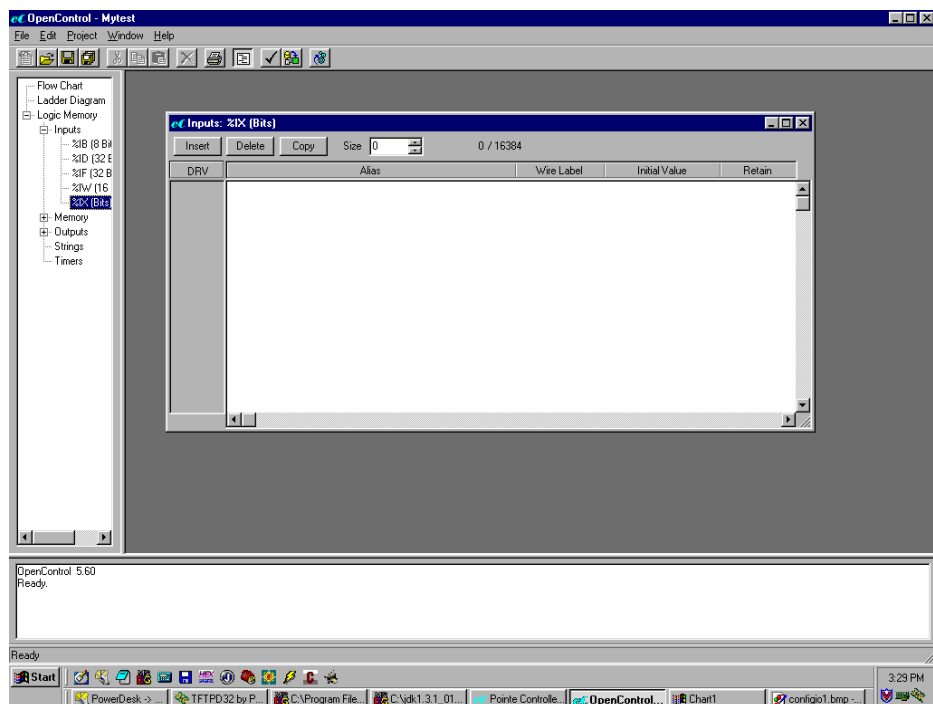
After we have defined all of the I/O, we will set up the timer that drives the project

Lastly, we will need to define a few reusable data variables for use in our internal calculations, but we will address those as we add each of the progress indicators.

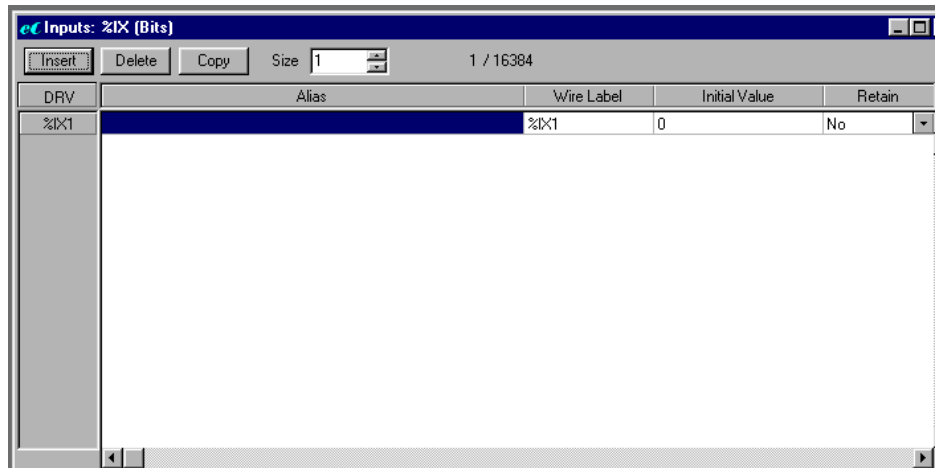
3.2.1 Defining input bits

First, to define the input tags:

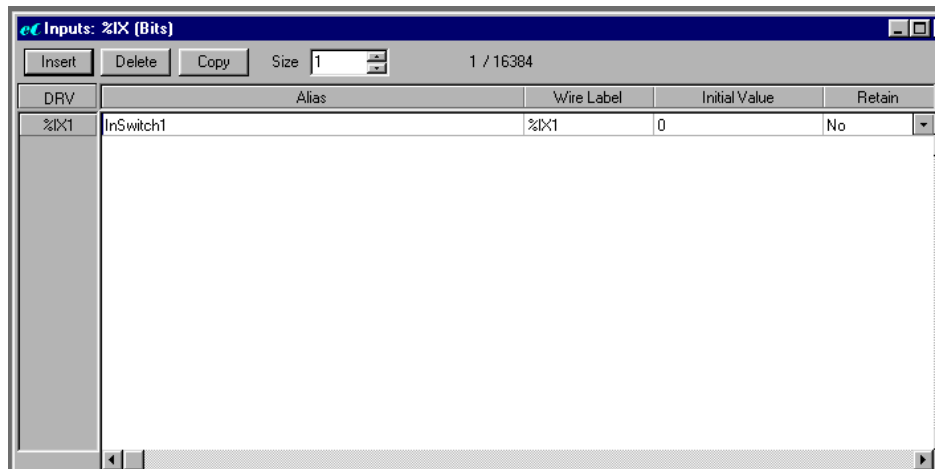
1. In the project workspace pane on the left, expand the list to show **Logic Memory > Inputs** and then double-click **%IX (Bits)**. This opens the input tag editor window.



2. Create a table entry for the first input tag (%IX1), either by clicking the **Insert** button or by incrementing the **Size** counter up to 1.

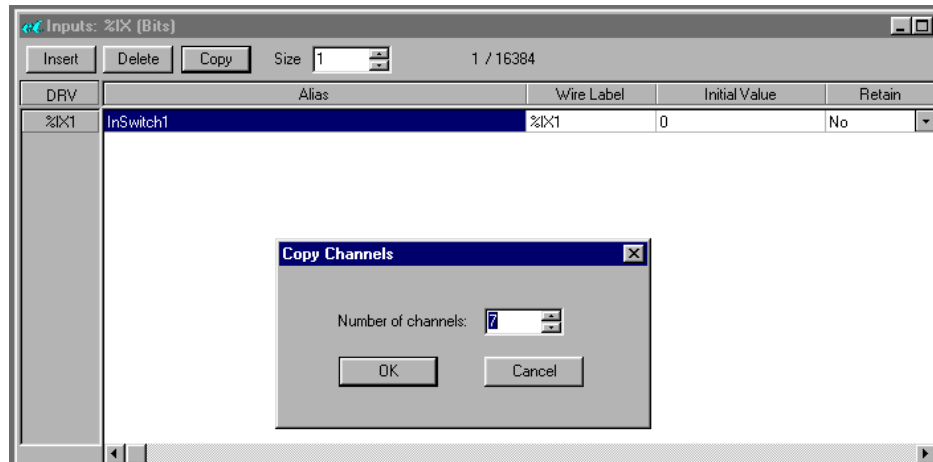


3. In the **Alias** field, enter the name of the input tag: InSwitch1. (You will need to click in the field before you can type in it.)

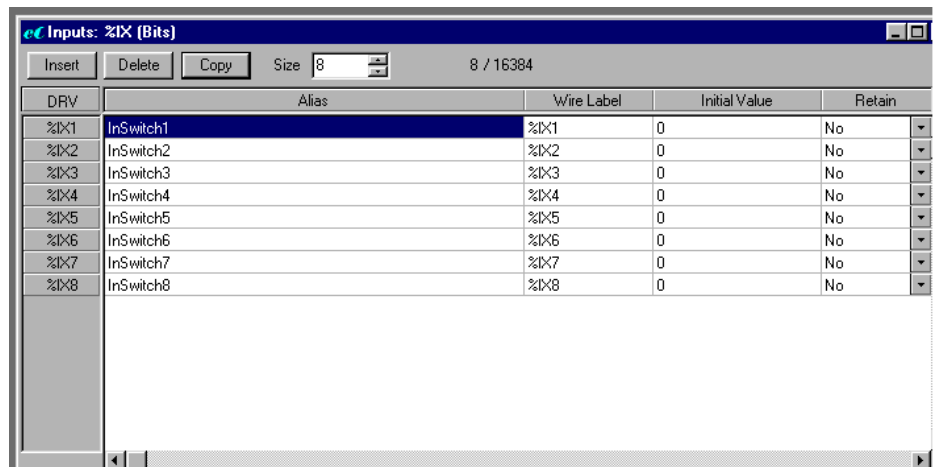



Now go ahead and define the rest of the tag names for the input module, even though we're not going to use them in this tutorial project. We've already used the name InSwitch1 for the first switch, so let's name the rest of the inputs InSwitch2 through InputSwitch8.

- Click the **Copy** button. The Copy Channels dialog box will appear.



- Increment the **Number of channels** up to 7 and click **OK**. Seven more input tags will be defined in numerical order, copied from the original InSwitch.



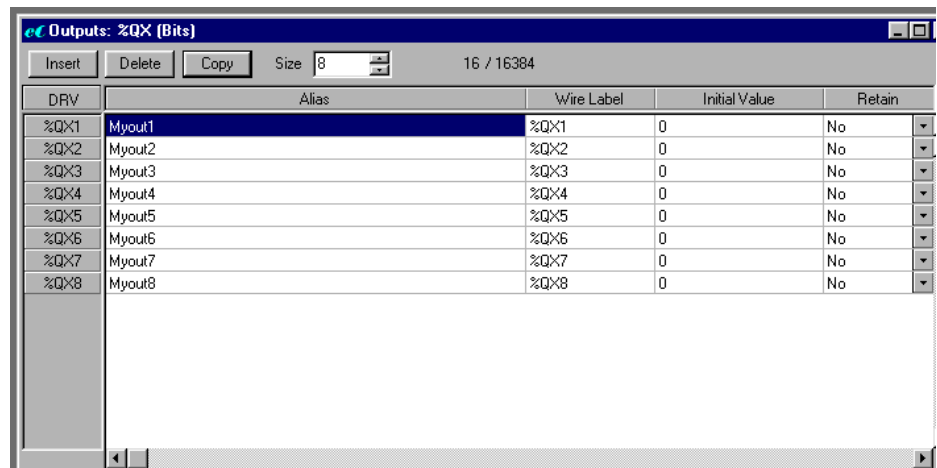
- Close the input tag editor window using the  button in the corner. When asked to save changes, click **Yes**.

3.2.2 Defining output bits

Now let's define our output tags using the same basic procedure as we used to define our input tags:

- In the project workspace pane on the left, expand the list to show **Logic Memory > Outputs** and then double-click **%QX (Bits)**. This opens the output tag editor window.
- Create a table entry for the first output tag (%QX1), either by clicking the **Insert** button or by incrementing the **Size** counter up to 1.


3. In the **Alias** field, enter the name of the output tag: Myout1. You will need to click in the field before you can type in it.
4. Click the **Copy** button. The Copy Channels dialog box will appear.
5. Increment the **Number of channels** up to 7 and click **OK**. Seven more input tags will be defined in numerical order, copied from the original Myout.



6. Close the output tag editor window using the  button in the corner. When asked to save changes, click **Yes**.

3.2.3 Defining memory tags

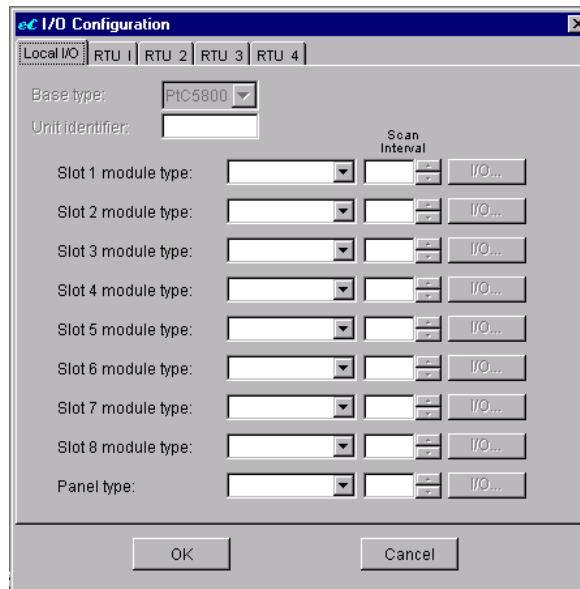
Lastly, we'll define the memory tags that will keep track of the state of the program:

1. In the project workspace pane on the left, expand the list to show **Logic Memory > Memory** and then double-click **%MX (Bits)**. This opens the output tag editor window.
2. Create a table entry for the first string (%MX1), either by clicking the **Insert** button or by adjusting the **Size** counter up to 1.
3. In the **Alias** field, enter the name of the input tag: rightOn. You may need to click in the field before you can type in it.
4. Close the memory tag editor window using the  button in the corner. When asked to save changes, click **Yes**.

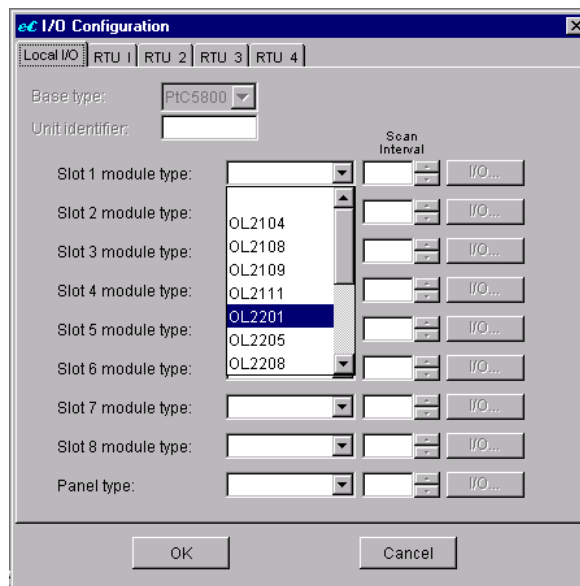
3.3 Associating Tags with I/O Points

At this point, we have tag-names defined. However, we haven't configured our I/O and haven't associated our tag-names with particular I/O points. Let's do that now:

1. From the **Project** menu, choose **Configure I/O**. The I/O Configuration window will appear.

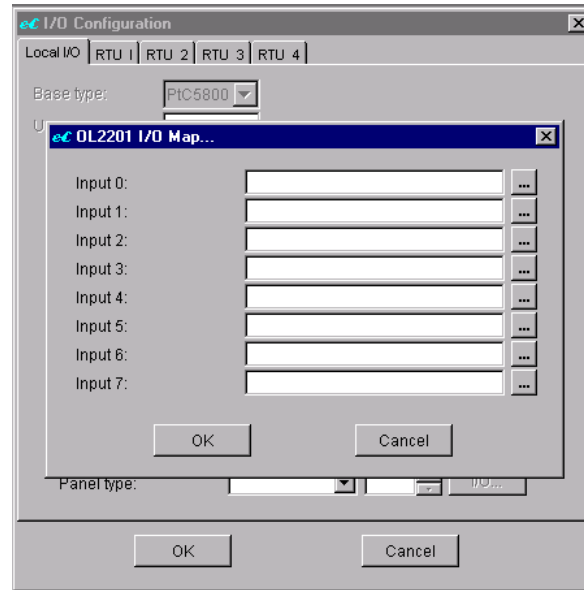


2. Next to **Slot 1 module type**, click the drop-down menu and select **OL2201**. This configures the OL2201 module in slot 1 of the Pointe Controller base unit.



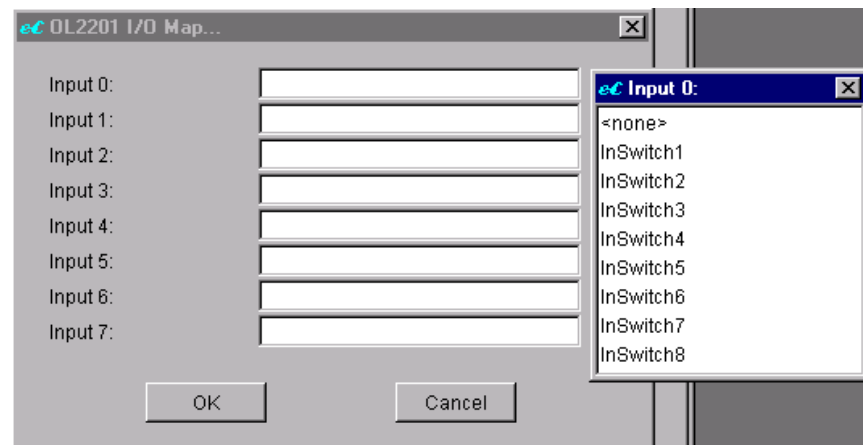
NOTE: Configuring slot 1 activates the **I/O** button to the right of the slot.

3. Click the **I/O** button for slot 1. The OL2201 I/O Map window will appear.

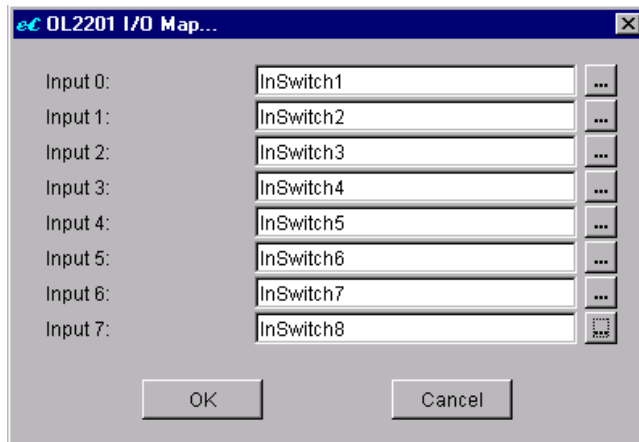


At this point of our project development, we have defined input tag names and selected an input module. What we haven't done is associate each input tag name with a particular point on the input module. That is what we will do now.

4. To the right of Input 1, click the **...** button. A list of available input tags will appear.



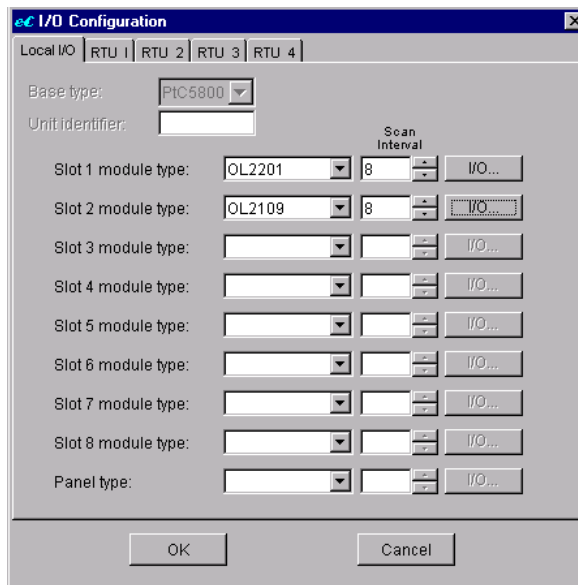
5. From the list of available input tags, select **InSwitch1**.
6. Repeat steps 4 and 5 for each of the remaining seven inputs, associating InSwitch2 to Input 2, InSwitch3 to Input 3, and so on.



7. Click **OK** to close the OL2201 I/O Map window.

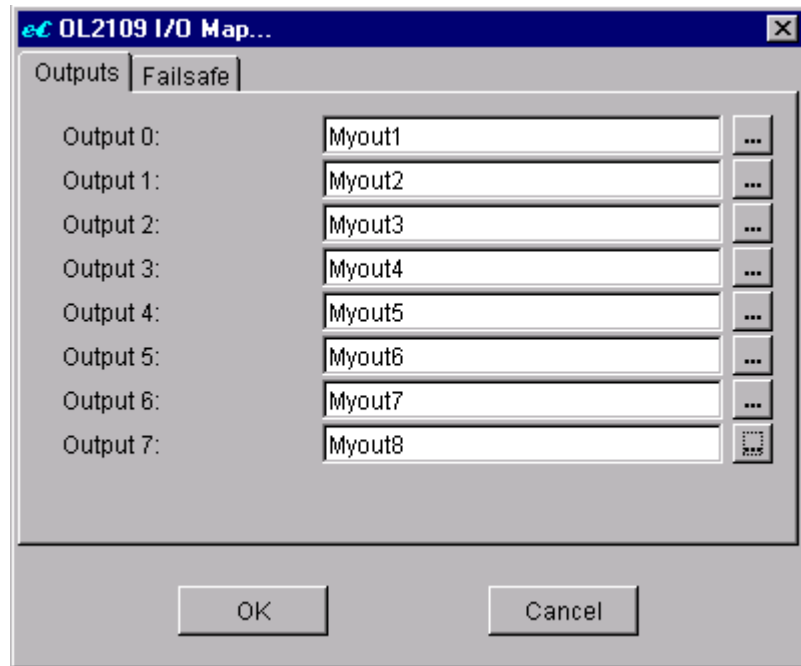
Now repeat the procedure to associate the eight output tags with the OL2109 module in slot 2:

8. Next to **Slot 2 module type**, click the drop-down menu and select **OL2109**. This configures the OL2109 module in slot 2 of the Pointe Controller base unit.



9. Click the **I/O** button for slot 2. The OL2109 I/O Map window will appear.
10. To the right of Output 1, click the **...** button. A list of available output tags will appear.
11. From the list of available output tags, select **MyOut1**.

- Repeat steps 10 and 11 for each of the remaining seven outputs, associating MyOut2 to Output 2, MyOut3 to Output 3, and so on.

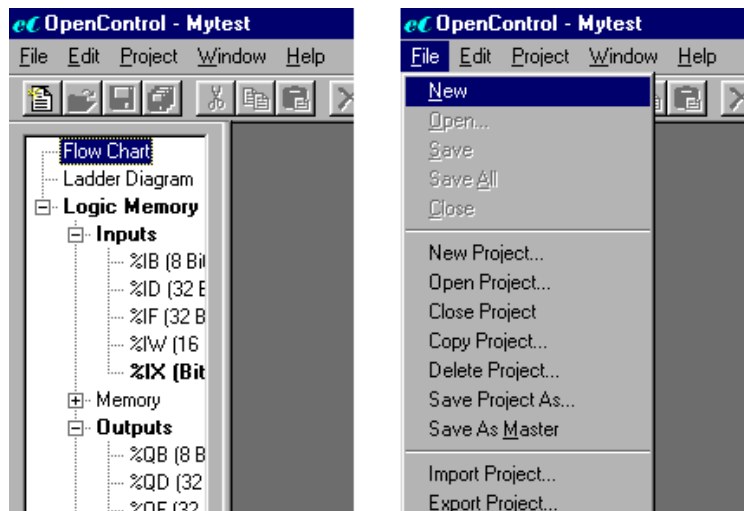


- Click **OK** to close the OL2109 I/O Map window.
- Click **OK** to close the I/O Configuration window.

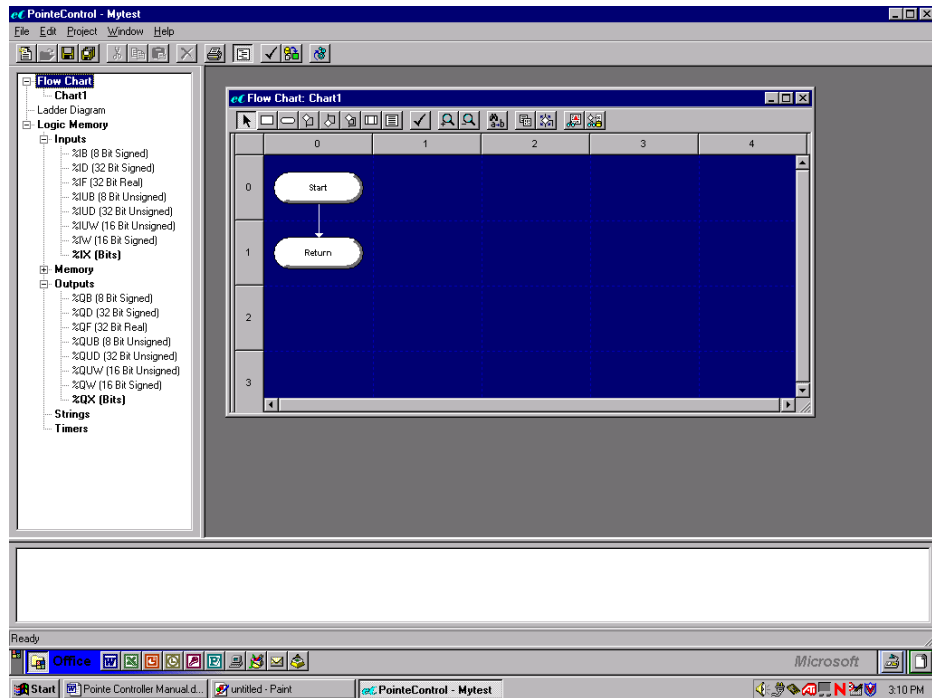
3.4 Creating Your First Flow Chart

We're going to do our first application with a flow chart. To create a new flow chart:

1. In the project workspace window on the left, select **Flow Chart**. Then from the **File** menu, choose **New**.



When you do this, a new flow chart will be created and added to the project tree.

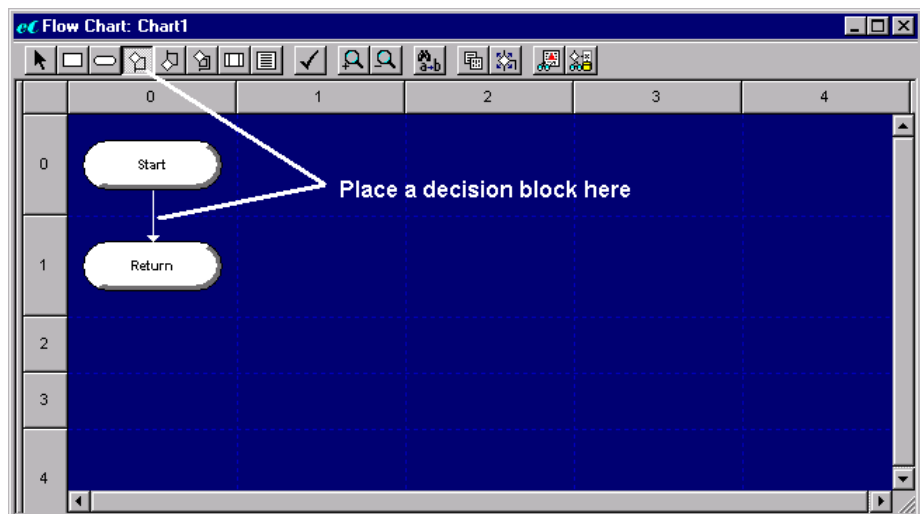


We could give our flow chart a name right now. In most cases, you would immediately name your flow chart based on the function it performs in your application. Typical names include "Gantry 1," "IPA Tank," "Purge Cycle," and so on. We're going to wait to give our program a name and use the default name "Chart1" for right now.

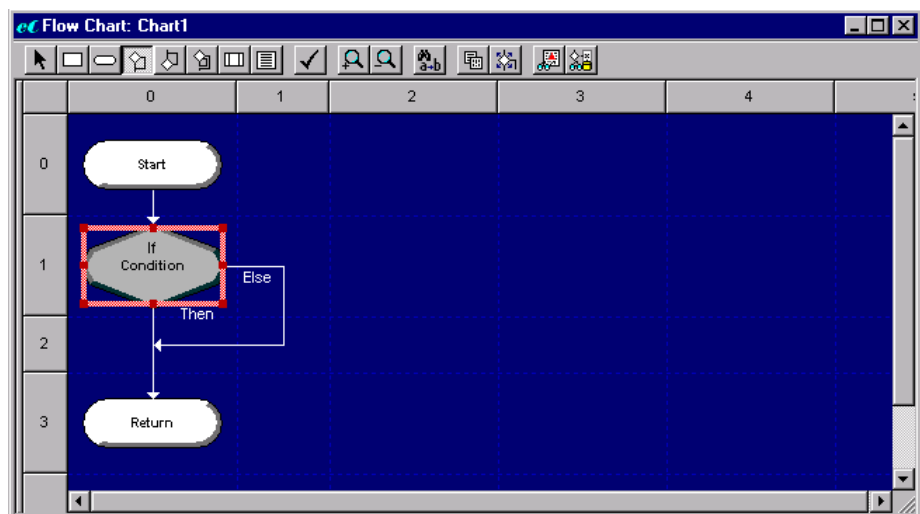
As you can see on your screen right now, a flow chart with nothing but a "Start" block and a "Return" block comes up on your screen. We're ready to enter our program.

Since we want to flash the outputs if Switch 1 is on, lets start with adding a decision block based on Switch 1's state:

2. Click the **Decision Block** tool on the toolbar, and then click on the flow line between the existing Start and Return blocks.



A generic decision block will be placed in your flow chart.



3. Double-click on the decision block to open its associated Block Properties window.

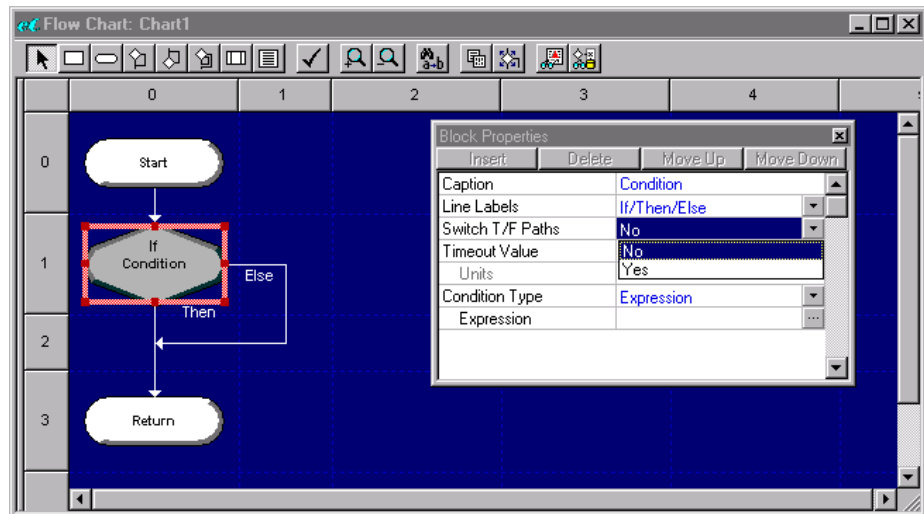
The Block Properties window is where you define what the block will do. As you can see, there are a number of properties that you can configure.

Caption determines the text description of the block. This is useful in documenting the project, but we shall leave it unchanged for now and come back to it later.

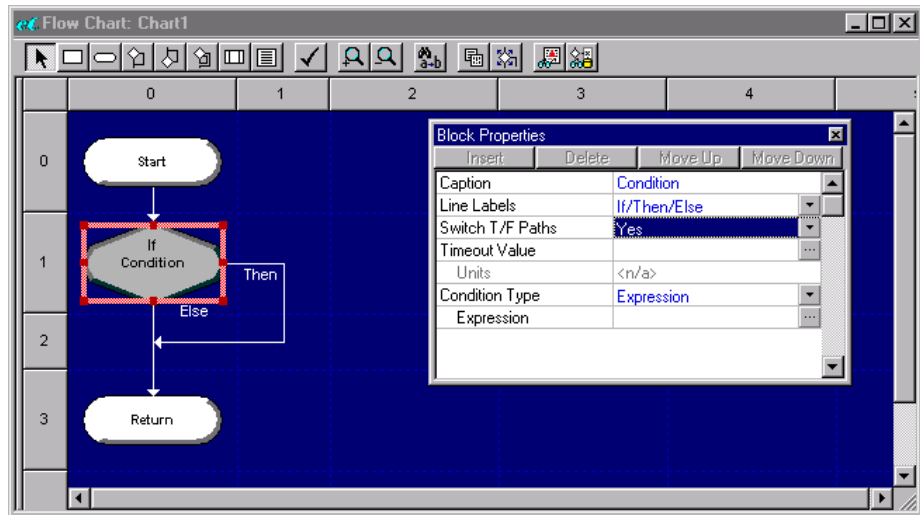
Line Labels determines the type of decision to be made by the block. We shall use the standard If/Then/Else labels.

Switch T/F Paths determines in which direction the logic flow will proceed, depending on whether the decision resolves as True or False. In the default No setting, True (Then) continues downward while False (Else) branches to the right. In the Yes setting, False (Else) continues downward while True (Then) branches to the right. This choice affects the overall readability of the flow chart and how the rest of the blocks will be oriented. For this quickstart project, let's switch the paths:

4. In the Block Properties window, click the **Switch T/F Paths** drop-down menu and select **Yes**.



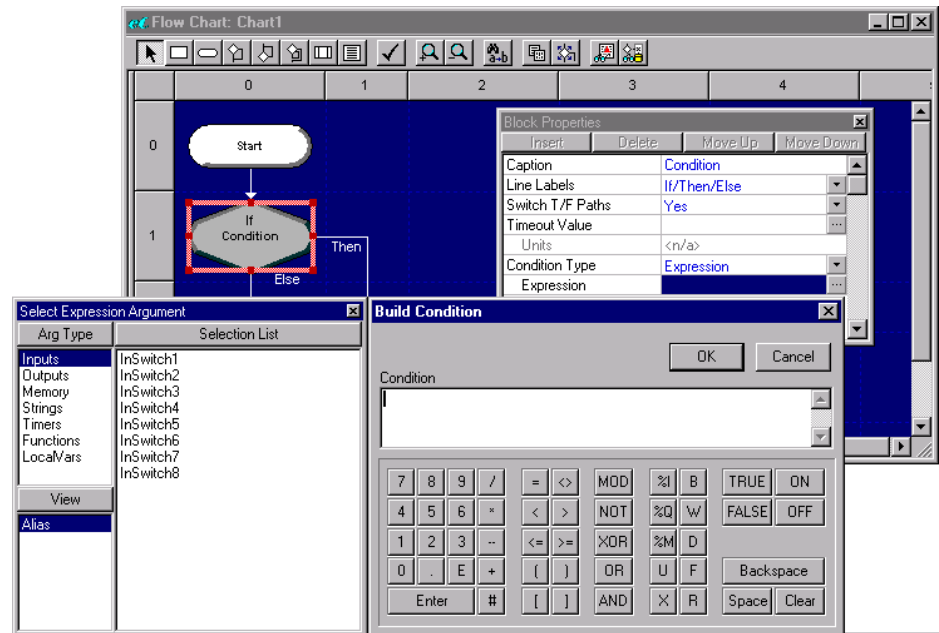
When the property is changed, the Then and Else paths are switched.



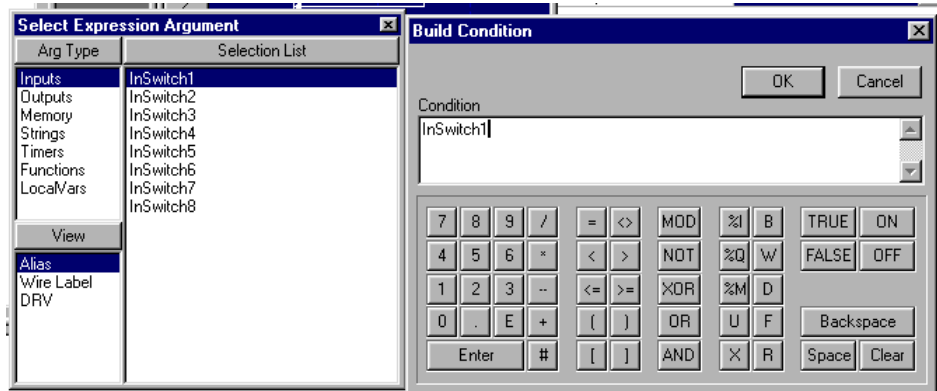
Timeout Value is an optional override that we won't use here.

Condition Type is an Expression (the default). We want to define our expression as "If InSwitch1 = ON":

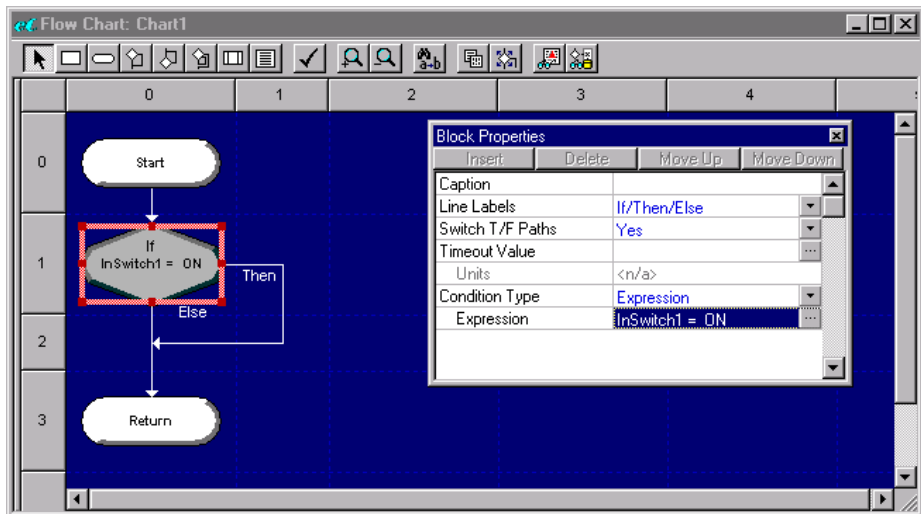
- To the right of the Expression property, click the button. The Build Condition window will appear.
- In the Arg Type list on the left, select **Inputs**. A list of available inputs will appear under Selection List.



- In the Selection List, select **InSwitch1**. The InSwitch1 tag will be added to the Condition pane.



8. Click the = button to add "=" to the condition.
9. Click the **ON** button to add "ON" to the condition.
10. Click **OK** to close the Build Condition window and enter the condition in the Expression property.

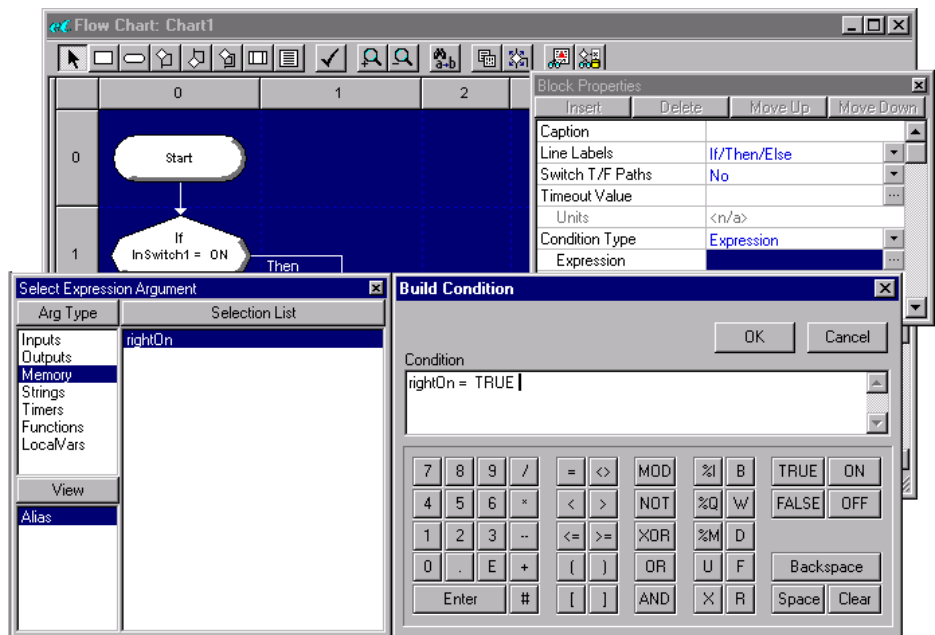


11. Close the Block Properties window.

3.5 Inserting a Second Decision Block

So far, all we've done is check to see if InputSwitch1 is on. If it is on, we want to flash the first two output LEDs out of phase. To do so, we turn the first output on and the second off for a period of time, then turn the first output off and the second on for a time period. Our flow chart will use the flag that we previously defined, rightOn, to keep track of which state we are in.


1. Click the **Decision Block** tool on the toolbar, and then click on the Then branch to the right of your "InSwitch1 = ON" decision block. A new decision block will be inserted.
2. Double-click on the new block to open its associated Block Properties window.
3. Define the block's Expression as "rightOn = TRUE," as described previously. Remember that rightOn is a Memory tag rather than an Input tag.

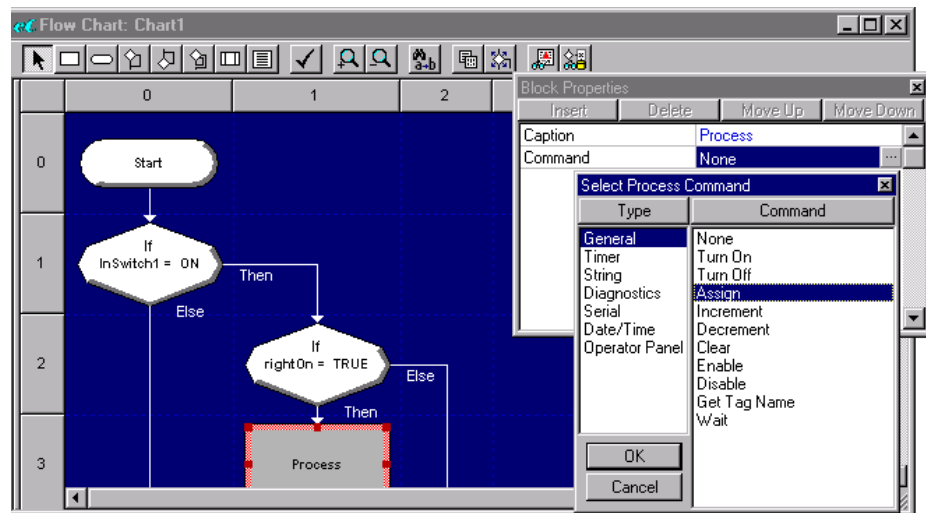



4. Close the Block Properties window.

3.6 Assigning Outputs

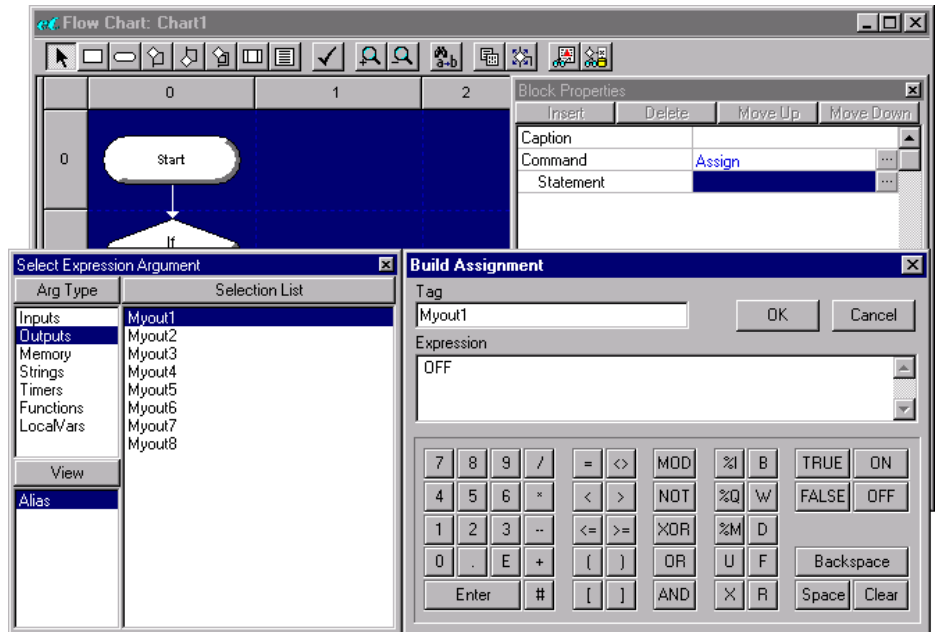
Now we are going to add the first of the two output patterns — turn off the first output LED (MyOut1) and turn on the second output LED (MyOut2).

1. Click the **Process Block** tool on the toolbar, and then click on the Then branch below your “rightON = TRUE” decision block. A new Process block will be inserted.
2. Open the Process block’s Block Properties window, as described previously.
3. To the right of the Command property, click the  button. The Select Process Command window will appear.
4. In the Type list, select **General**. A list of general commands will appear.
5. In the Command list, select **Assign**.

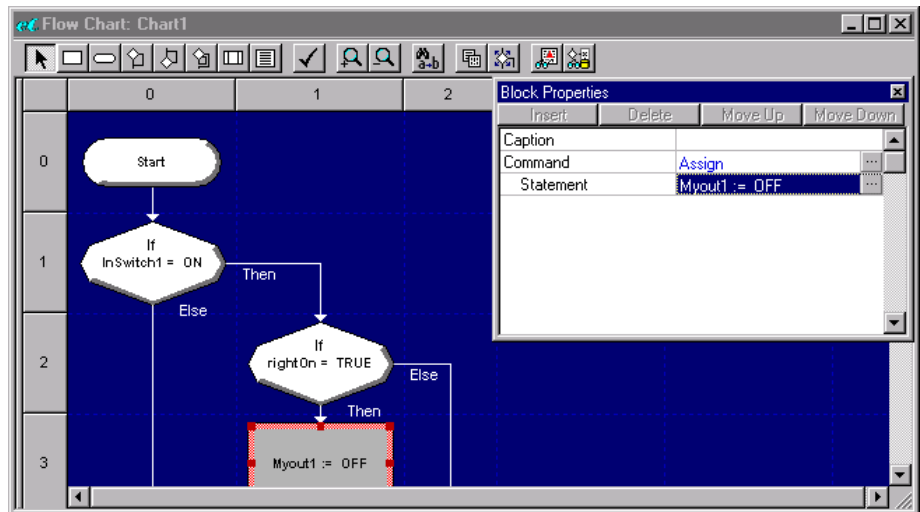


6. Click **OK** to close the Select Process Command window and enter the Assign command in the Command property.
7. You will see that a Statement sub-property is added to the Command property.
8. To the right of the Statement sub-property, click the  button. The Build Assignment window will appear.
9. Select the **MyOut1** output tag, as described previously.

10. Move the cursor to the Expression field, either by clicking in it or by pressing the Tab key.
11. Click the **OFF** button to add "OFF" to the expression.

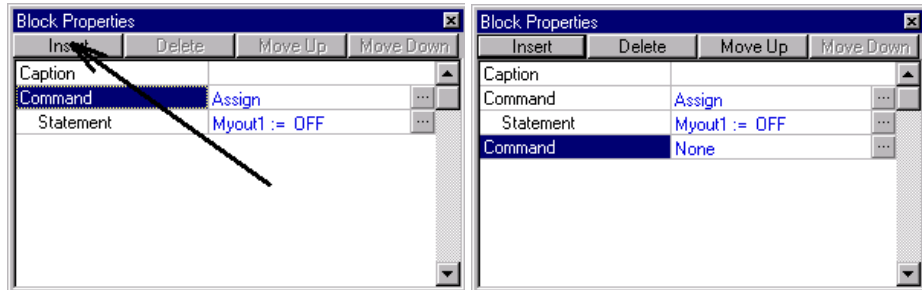


12. Click **OK** to close the Build Assignment window and enter the assignment in the Statement sub-property.

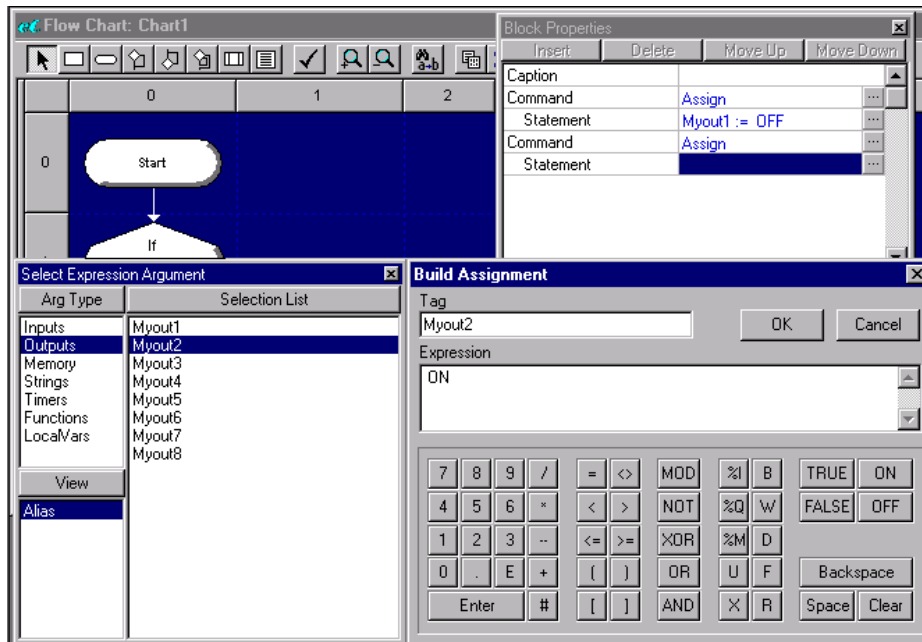


13. Now let's add a second Assign command to this same Process block.

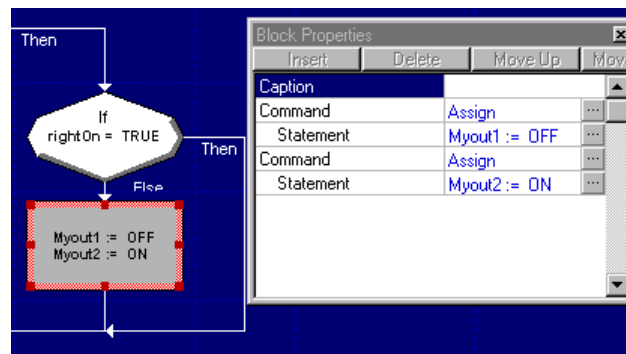
- Click on and highlight the existing Command property, and then click the **Insert** button. A second Command property will be inserted.



- Define the second Command property as "MyOut2 = ON," as described previously.

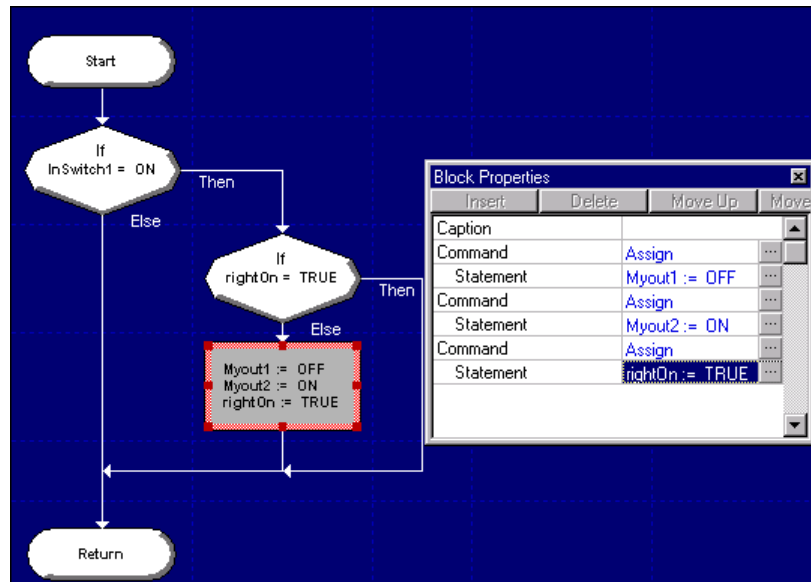


Now we have two Assign commands in the Process block, as shown below.



Let's add just one more statement to set rightOn to TRUE. This will force the decision block to take the other branch on the next scan through. In that branch, we'll set the output states to the opposite values. That will create the flashing effect that we are looking to achieve.

16. Insert a third Command property, as described previously.
17. Define the third Command property as "rightOn = TRUE," as described previously. Remember that rightOn is a memory tag rather than an output tag.

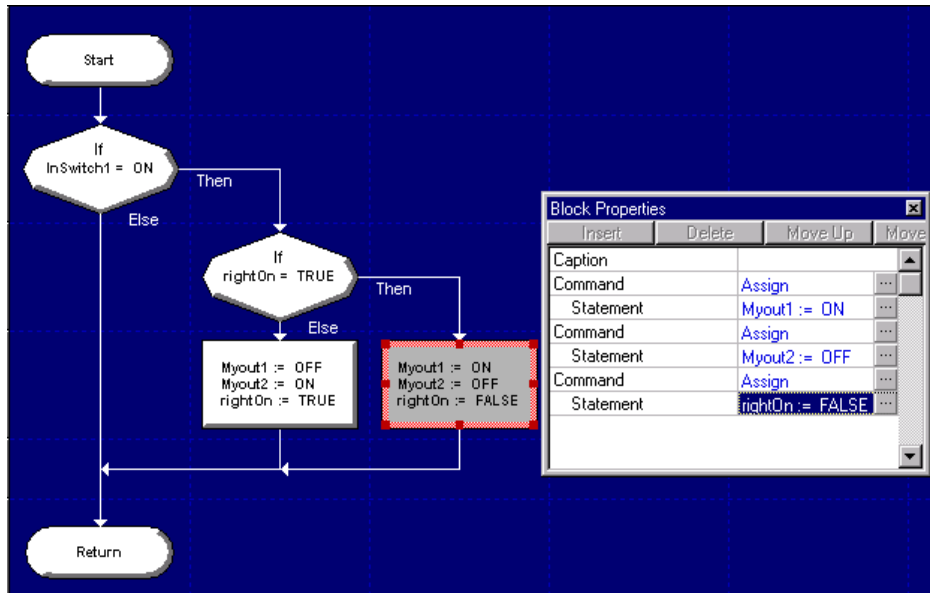


18. Close the Block Properties window.

Now we need to add the second of the two output patterns — turn on MyOut1, turn off MyOut2, and set rightOn to FALSE.

19. Insert a new Process block in the Then branch to the right of "rightOn = TRUE" decision block.
20. Open the Process block's Block Properties window.
21. Using the same procedure as before, define three Assign commands in the Process block:
 - "MyOut1 = ON"
 - "MyOut2 = OFF"
 - "rightOn = FALSE"

The Process block should appear as shown below.



22. Close the Block Properties window.

Looking at our flow chart, we can now see that if the toggle switch (InSwitch1) is on, then the output LEDs (MyOut1 and MyOut2) will flash back and forth. As soon as the chart reaches the end, it returns to the start and scans through again.

If the toggle switch is off, then the output LEDs freeze in their last state until the switch is on again.

3.7 Adding a Time Delay

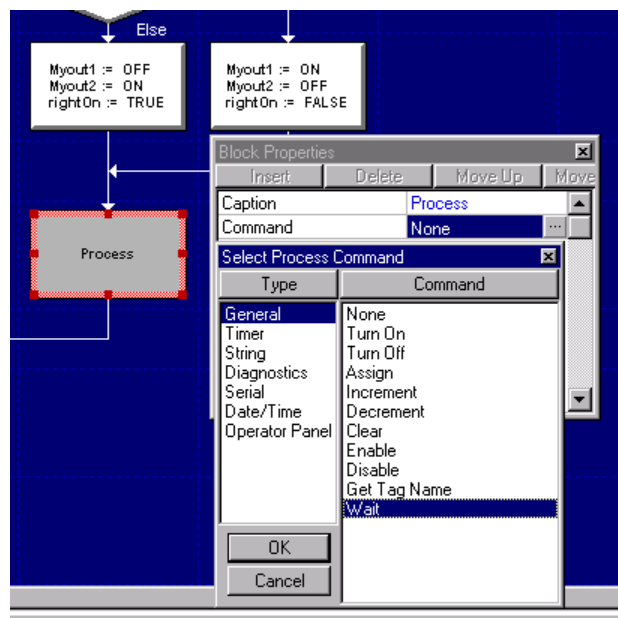
Unfortunately, when the toggle switch is on, the output LEDs flash too rapidly for us to see. We need to put in a time delay of 500 milliseconds — just enough to slow the flashing to an observable speed.

1. Insert a new Process block after the two Process blocks that you defined earlier.

NOTE: If you place the block in the wrong place, there are two ways to fix it. The first way is to simply delete the block and insert it again. (For more information on deleting Flow Chart blocks, see page 143.)

The second way is to click on and drag the block to the correct position in the flowchart. The chart will automatically redraw itself to incorporate the block wherever you place it.

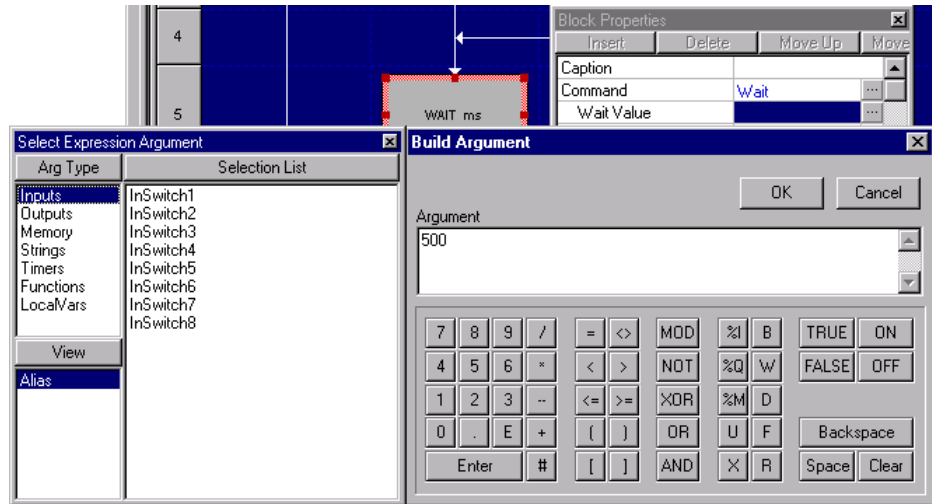
2. Open the new block's Block Properties window.
3. Set the Command property to **Wait**.



A Wait Value sub-property will be added.

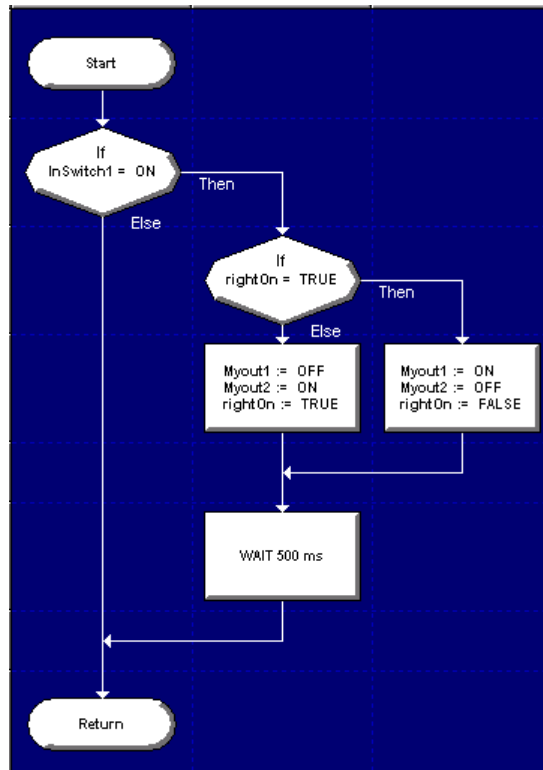
4. To the right of the Wait Value sub-property, click the button. The Build Argument window will appear.

- In the Argument field, enter a numerical value of **500**.



- Click **OK** to close the Build Argument window and enter the argument in the Wait Value sub-property.
- Close the Block Properties window.

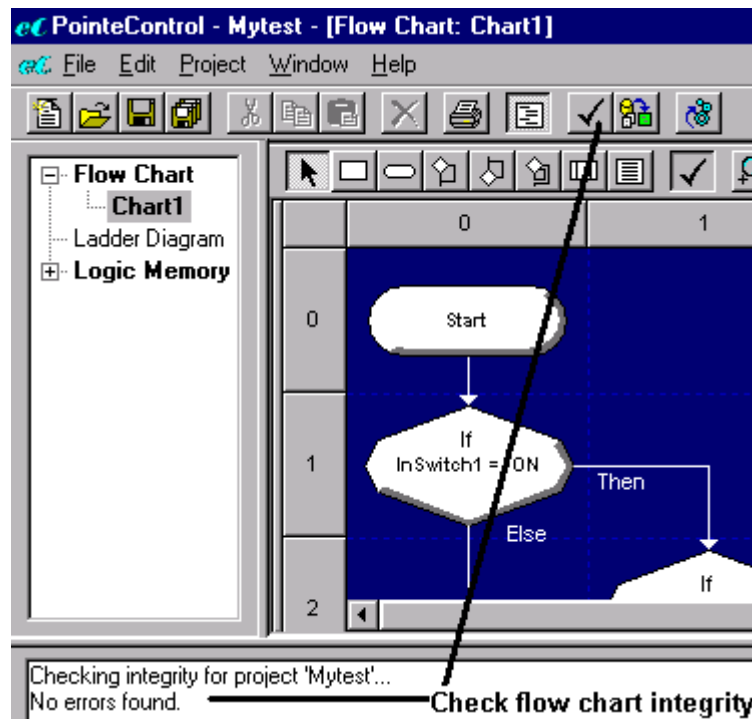
Your flow chart should now appear as shown below.



3.8 Checking the Chart Integrity

The final step in entry, or modification, of a flow chart is an integrity check. This integrity check will automatically check the chart for errors in the flow chart function blocks. It will not tell you if your flow chart logic is correct – it will only tell you if the statements have been entered properly.

To perform an integrity check, click the **Check Integrity** tool on the toolbar.



The results of the integrity check will appear in the *Message and Error Window Pane* (the box below the editor window). The figure on the right illustrates the message that you will get if your flow chart contains no statement errors. If an error is found, a message listing the error and the flow chart block where it is located, will appear in the *Message and Error Window*.

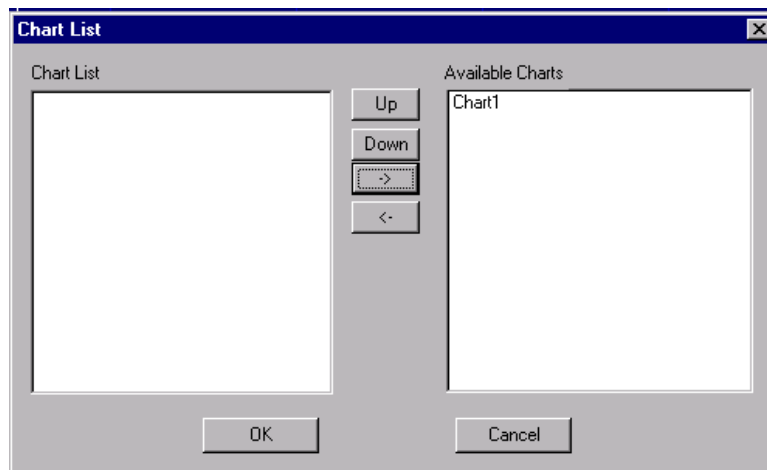
3.9 Building the Project Runtime

Once all of the flow charts have been created, we are ready to build the project runtime.

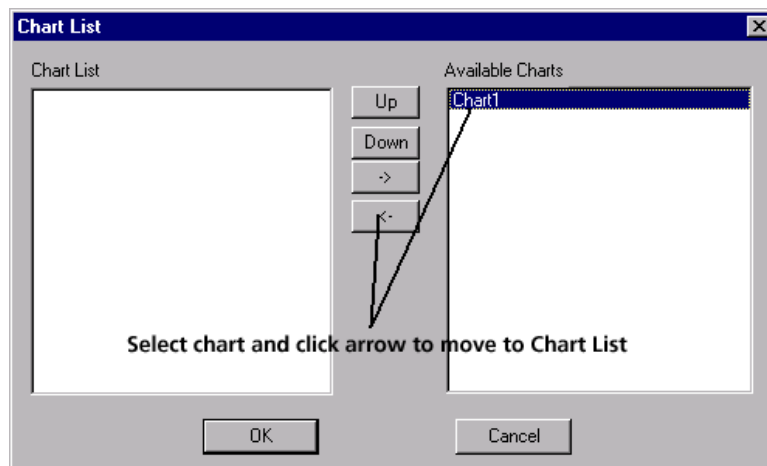
First, we must define the list of charts that make up the project. A project can be comprised of many flow charts and/or ladder diagrams. When a chart is created, it is not automatically added to the project list. This allows you to incrementally edit charts and keep optional charts within your development environment. When its time to build the project runtime, you can place the required charts in the project list.

To configure your project's chart list:

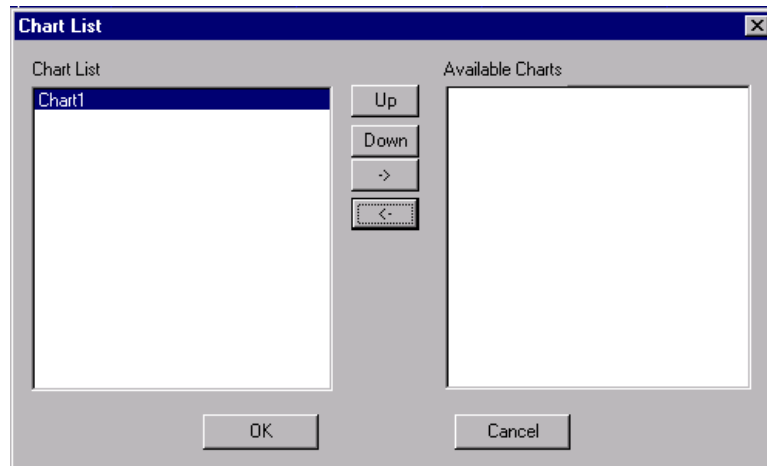
1. From the **Project** menu, choose **Configure Chart List**. The Chart List window will appear, showing all charts in the current project build and all other available charts. We've only created one chart. — it is listed as available.



2. In the Available Charts list, select **Chart1** and click the **<** button to move the chart to the active Chart List.

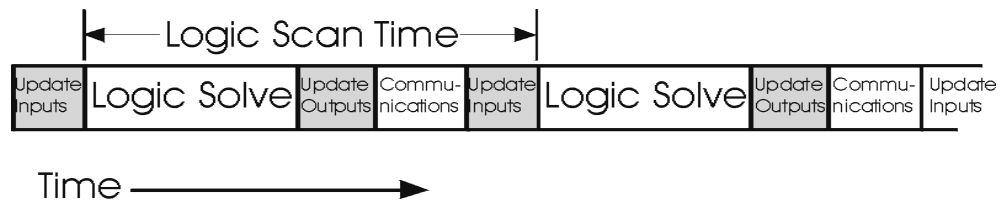


The result should be as shown on the right.

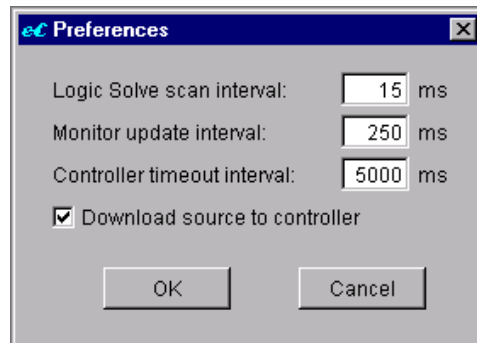


3. Click **OK** to close the Chart List window.

After you have configured the project’s chart list, you must define the scan parameters. Your project’s runtime operation actually occurs in a repetitive cycle of updating inputs, logic solve, updating outputs and communications.



4. From the **Edit** menu, choose **Preferences**. The Preferences window will appear.



Logic Solve scan interval is the time between the beginning of one logic solve pass and the next.

Monitor update interval is how frequently, the monitor, which we’ll use next for debug purposes, talks to the Pointe Controller to update its information. The lower the update interval number, the faster the monitor will react — but the more load it places on the Pointe Controller processor for handling communications.

Controller Timeout Interval is a watchdog timer for the chart execution. If the chart logic solve is greater than this value the project will stop.

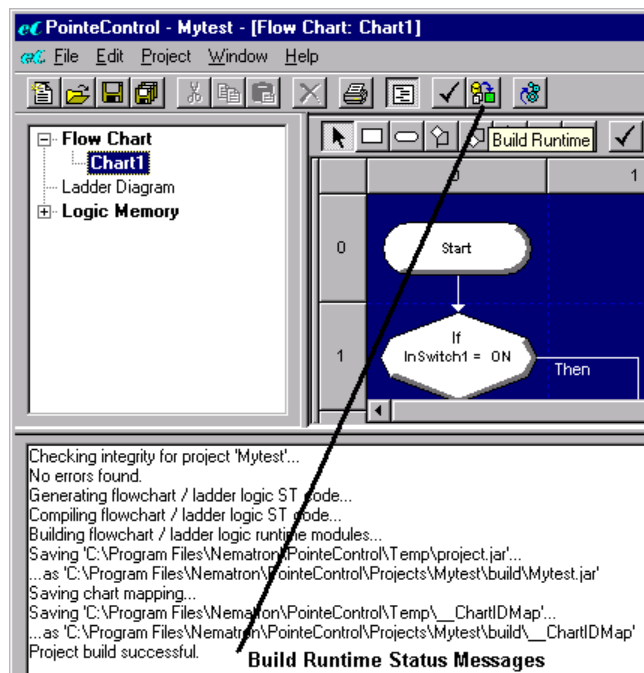
Download Source when checked will send the project source code to the Pointe Controller. If this is un-checked the source will not be present in the Pointe Controller and the Pointe Control Monitor will not be able to view or debug the application.

For our application, a 15 millisecond scan time setting and 250 millisecond monitor update interval are more than sufficient (actually, in most cases, you'll want to increase the monitor interval to 500 to 1000 milliseconds to reduce the CPU loading caused by the monitoring activities).

5. In the Logic Solve scan interval, enter **15** milliseconds.
6. In the Monitor update interval, enter **250** milliseconds.
7. Controller Timeout Interval to **5000** milliseconds.
8. Download Source must be checked.
9. Click **OK** to save your changes and close the Preferences window.

Now we can build our project runtime:


10. Click the **Build Runtime** tool on the toolbar.

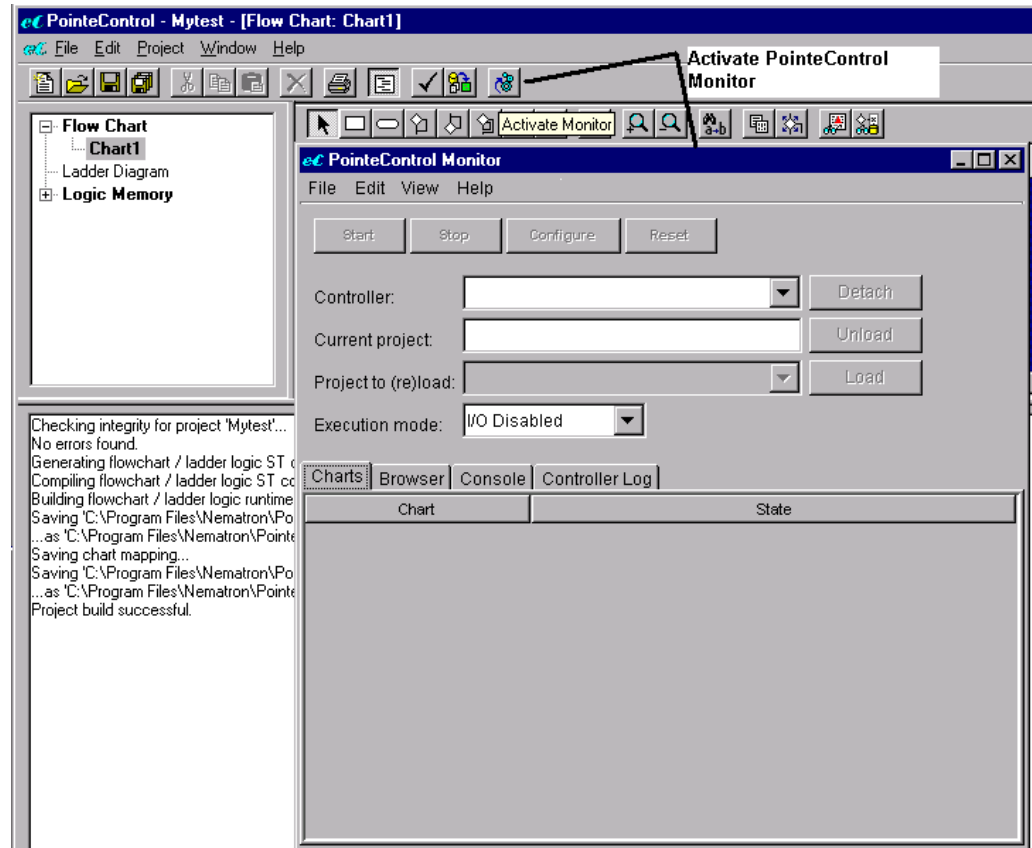


When the project builds, messages will come up in the Message Window, informing you of the progress of the build. The screen shot shown above illustrates a successful build. Two key message lines are the "No errors found" line and the "Project build successful" line. Most of the other messages are informative about the progress of the build.

3.10 Downloading and Running Your Program

Downloading a finished program to the Pointe Controller unit is performed via the PointeControl Monitor. The Monitor is launched separately from the development framework, either by choosing it from the Windows Start menu (**Start > Programs > PointeControl > Monitor**), or by “activating” it from within the framework itself.

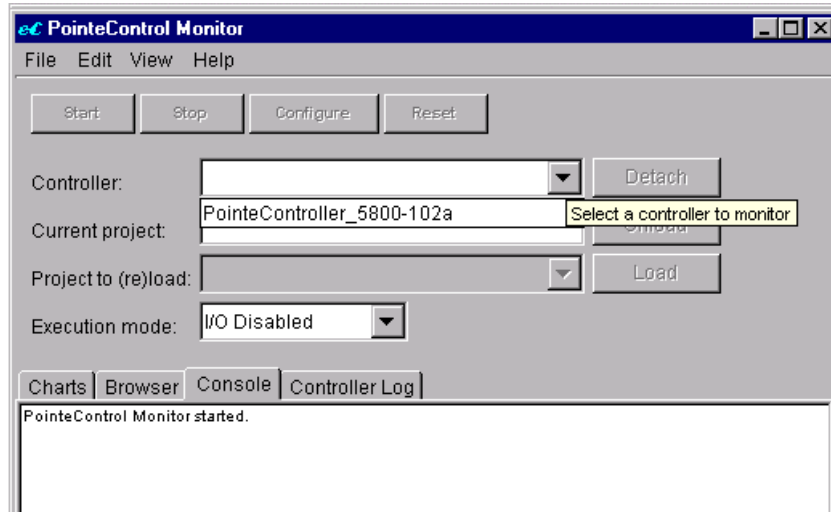
 To activate the monitor from within the framework, click the Activate Monitor button on the toolbar



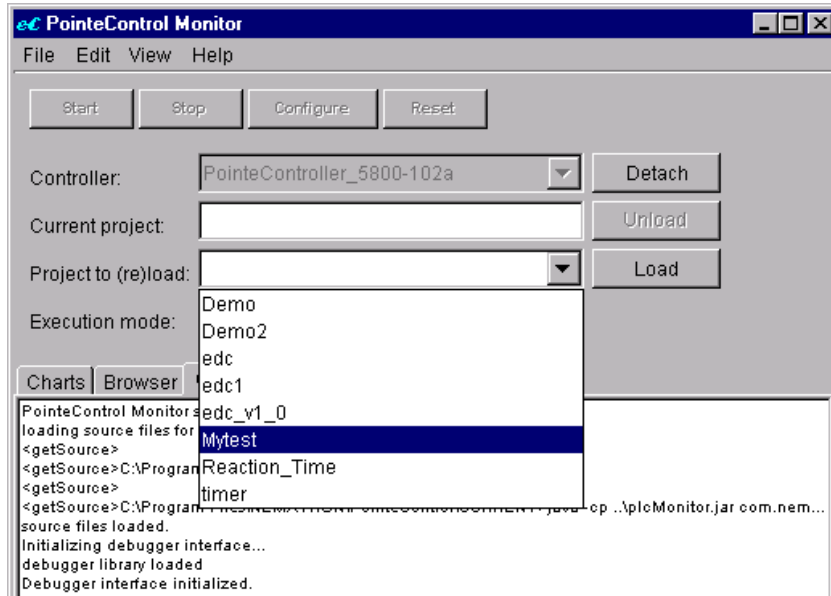
Alternately, you can choose **Activate Monitor** from the **Window** menu.

Once the PointeControl Monitor window is active, you can download your project runtime to the Pointe Controller unit:

1. Click the **Controller** drop-down menu and select the Pointe Controller unit from the listed devices.



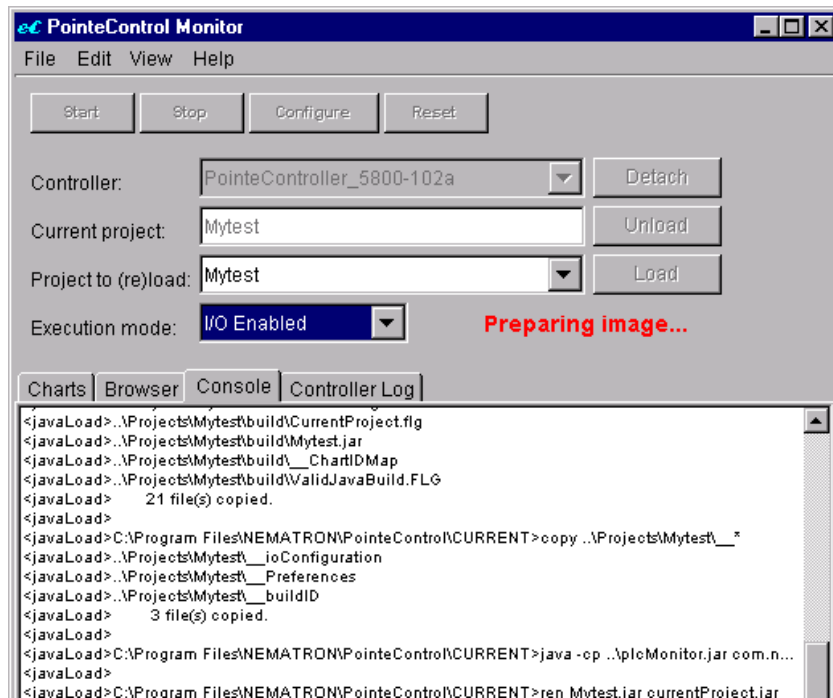
2. Click the **Project** drop-down menu and select your project runtime. If you have followed the examples given in the quickstart, your project runtime should be named **"Mytest."**



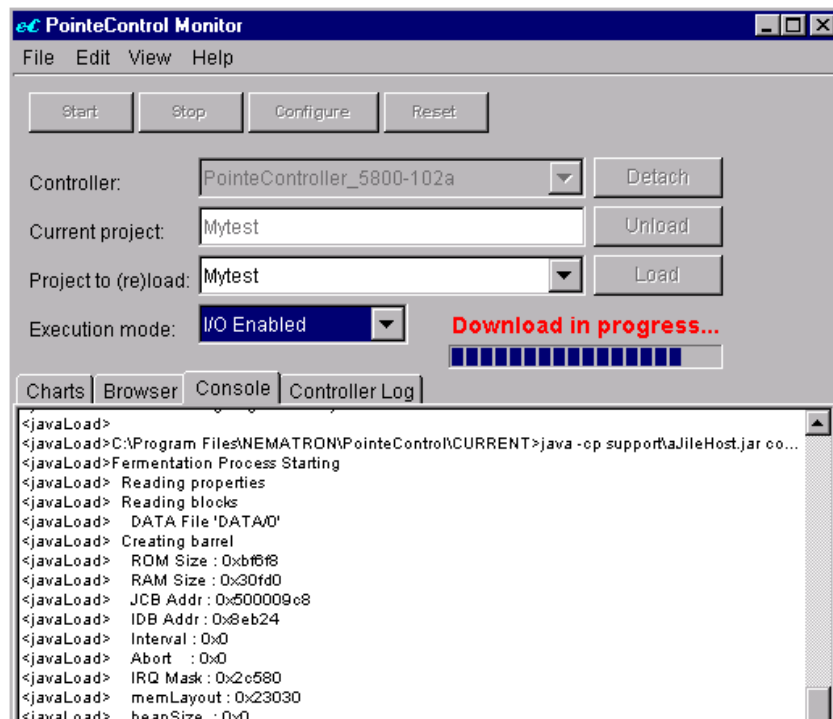
3. Click the **Load** button. Your PC will spend several moments preparing the download. (The exact time depends on the processing speed of your PC.)

TIP: You can get a detailed view of the download process by selecting the **Console** tab, as shown below. The console messages are generated by PointeControl Monitor as it prepares your project to run on the Pointe

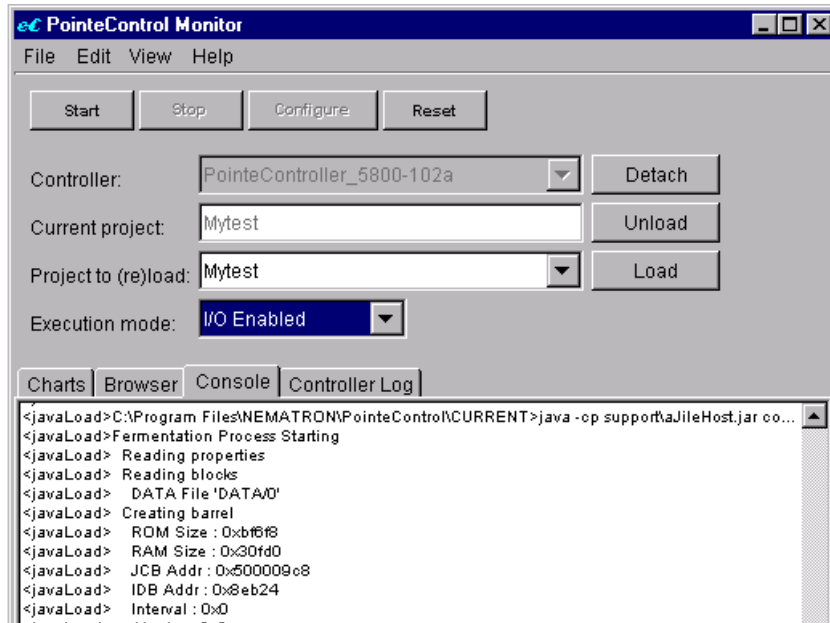
Controller’s Java-based processor. In most cases you can ignore these messages; they are useful only when troubleshooting a faulty download.



A progress bar in the Pointe Control Monitor window shows your project runtime is downloaded to the Pointe Controller unit.

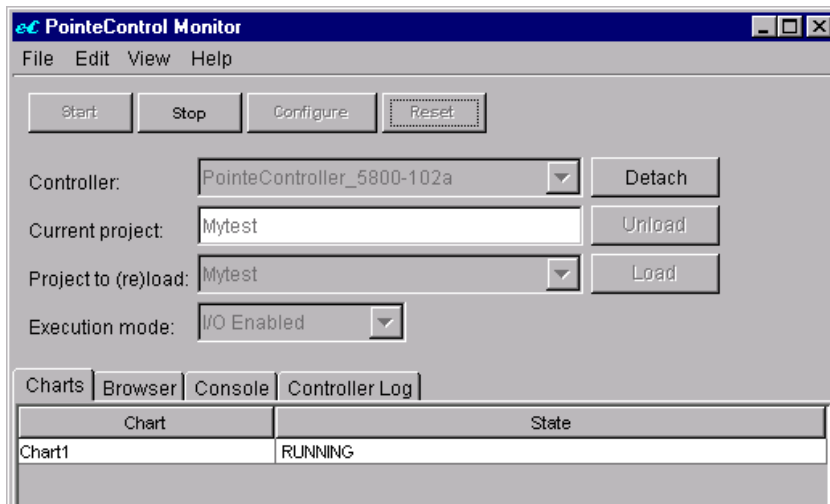


When the download is complete, the **Start** button will become active.



4. Click the **Start** button to run the project.
5. Select the **Charts** tab, as shown below.

The PointeControl Monitor window should show that your one chart (as configured in the Chart List) is running. The Pointe Controller unit is now executing your project as a real-time control program.

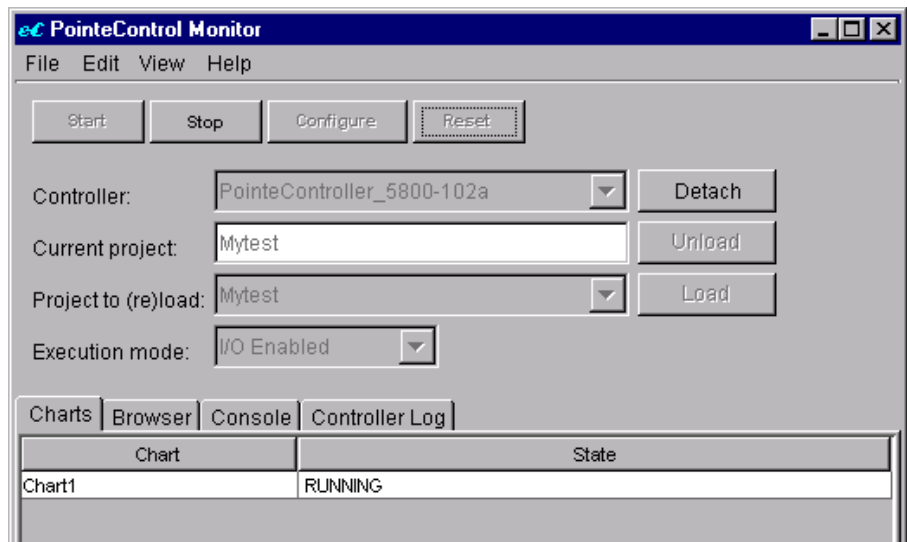


3.11 Monitoring Your Program While It Runs

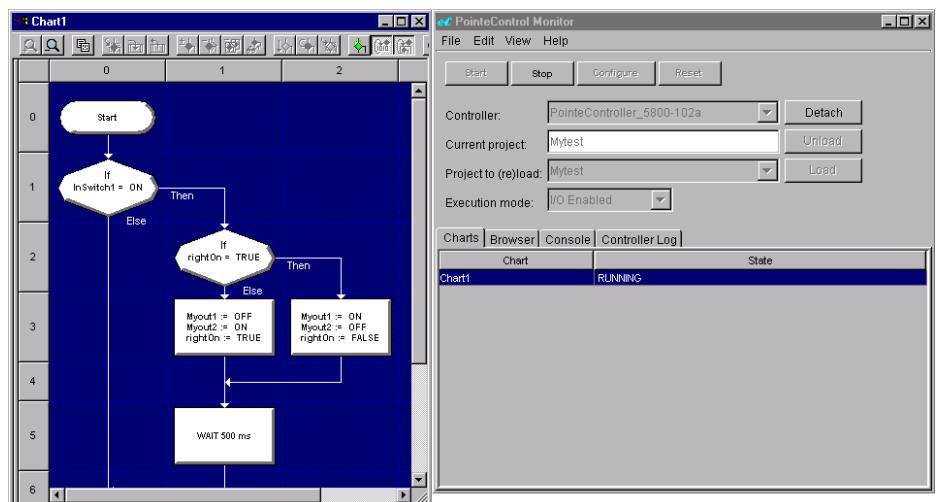
From your PC, you can monitor logic flow, view snapshots of tag values, change tag values, set breakpoints, single-step through your program, and get performance and loading information.

To monitor your program while it is running:

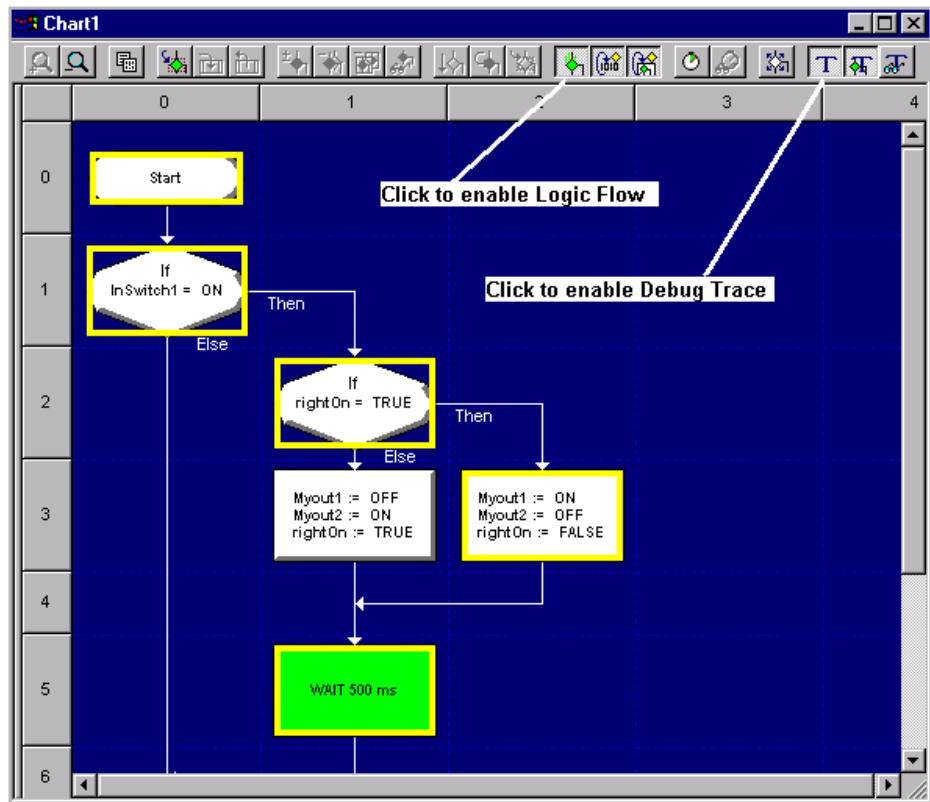
1. In the PointeControl Monitor window, click the **Charts** tab. A list of all currently running charts will be displayed.



2. In the Charts tab, double-click **Chart1**. The chart will be displayed in a separate monitor/debug window.



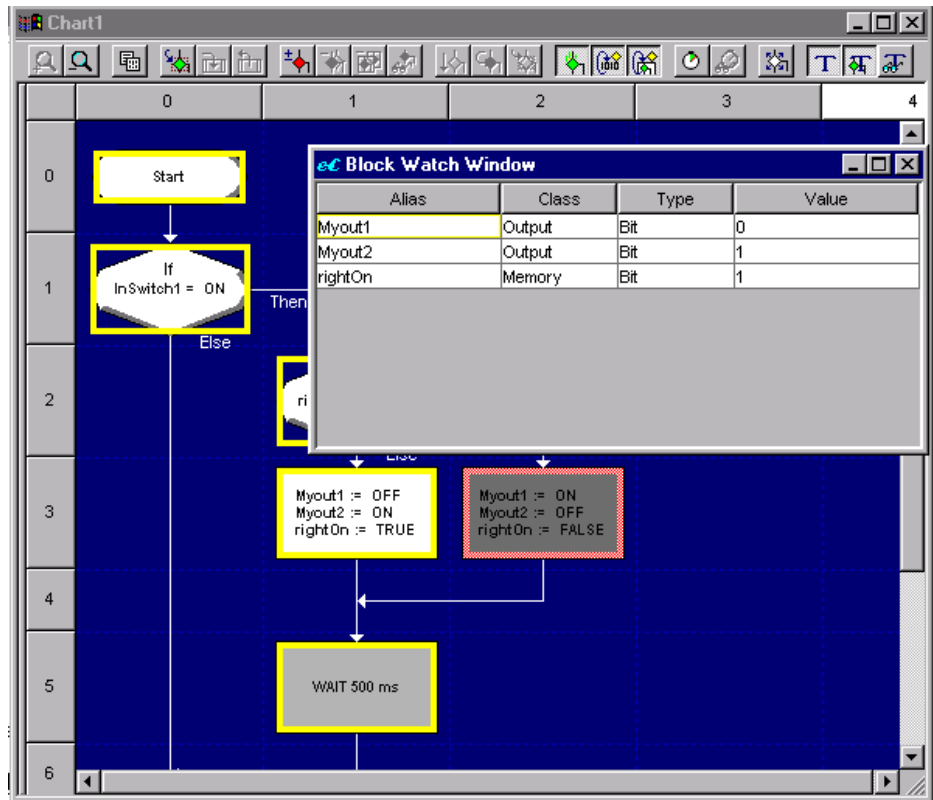
- In the Chart1 monitor/debug window, click the **Enable Logic Flow** and **Enable Debug Trace** tools on the toolbar.



You now have a snapshot of the program’s logic flow, highlighted in yellow and updated at a regular interval. The logic flow shown is the last 200 program blocks executed each time the monitor collects a snapshot. The snapshot is taken based on the **Monitor update interval** that you defined previously. (See “Building the Project Runtime” above.) If the interval is set to 100 milliseconds, then a snapshot will be taken every 100 milliseconds.

Flip the first switch on the OL2201 module and watch what happens. Depending on whether the switch is on or off, you should see the logic flow change. Watch the lights on the OL2109 output module. When the switch is in the on position, lights for module outputs 0 and 1 should alternate. You should be able to see this same alternating pattern in the logic flow.

4. Double-click on any of the flow chart blocks (except the Start and Stop blocks). A Block Watch window for that block will appear, showing all of the tags referenced by that block.



The Block Watch window shows all of the tags used in the block and the running value of each tag.

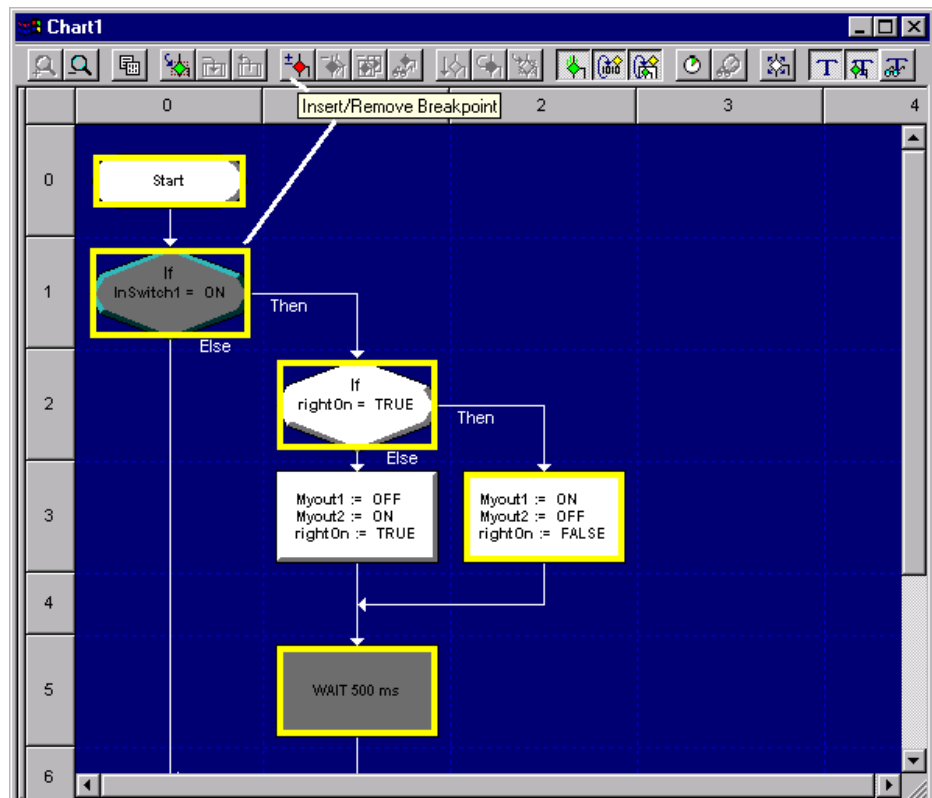
NOTE: The value shown for each tag is *not* the real-time value of the tag as the Pointe Controller is actually executing the block. It is the value of the tag at the time of last snapshot, which is generally more useful for debugging purposes.

5. Close the Block Watch window.

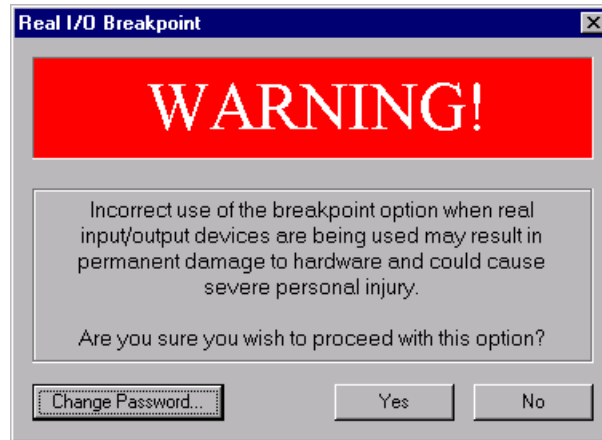
3.12 Setting Breakpoints

Sometimes you'll want to check if your program gets to a particular point in your chart. You may also want to step through the execution of the program one block at a time in order to debug your logic. This is achieved by setting breakpoints in the logic flow:

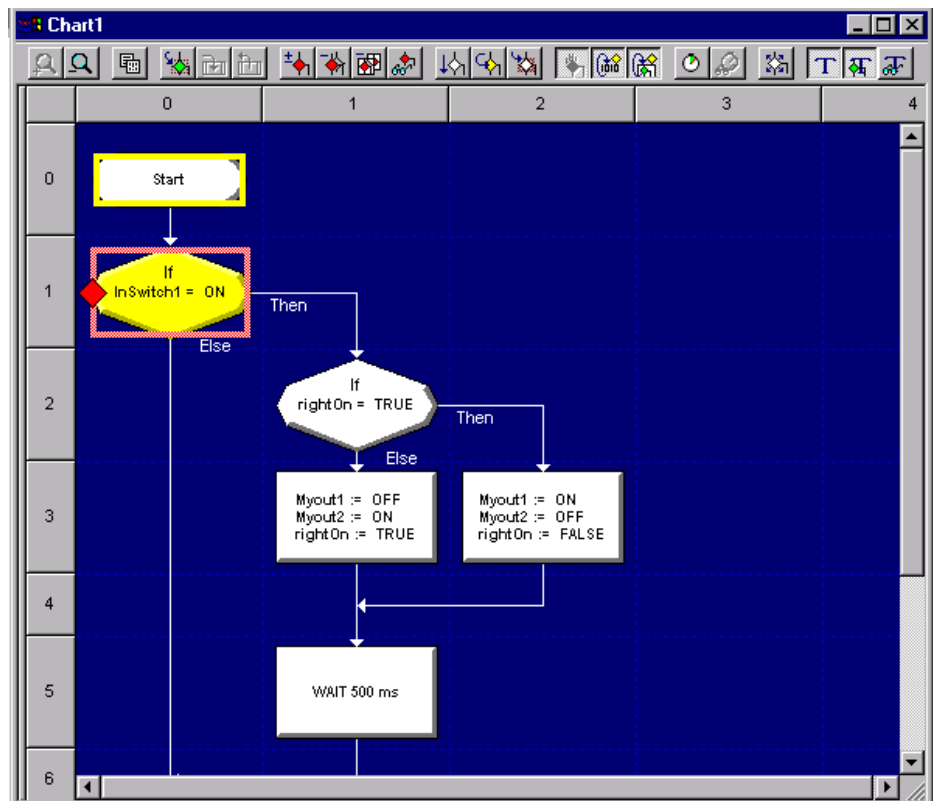
1. Make sure your program is running and Debug Trace is enabled. (See "Monitoring Your Program" above.)
2. Select a block in the flow chart. The block will be highlighted green.
3. Click the **Insert/Remove Breakpoint** tool on the toolbar.



When you do this, an alert window will pop up on your screen, warning you that stopping the flow of a running program could cause problems (if you were actually controlling a machine) and asking you to verify that you do in fact want to insert the break point.



4. Click **Yes**. A break point will be set on the selected block.

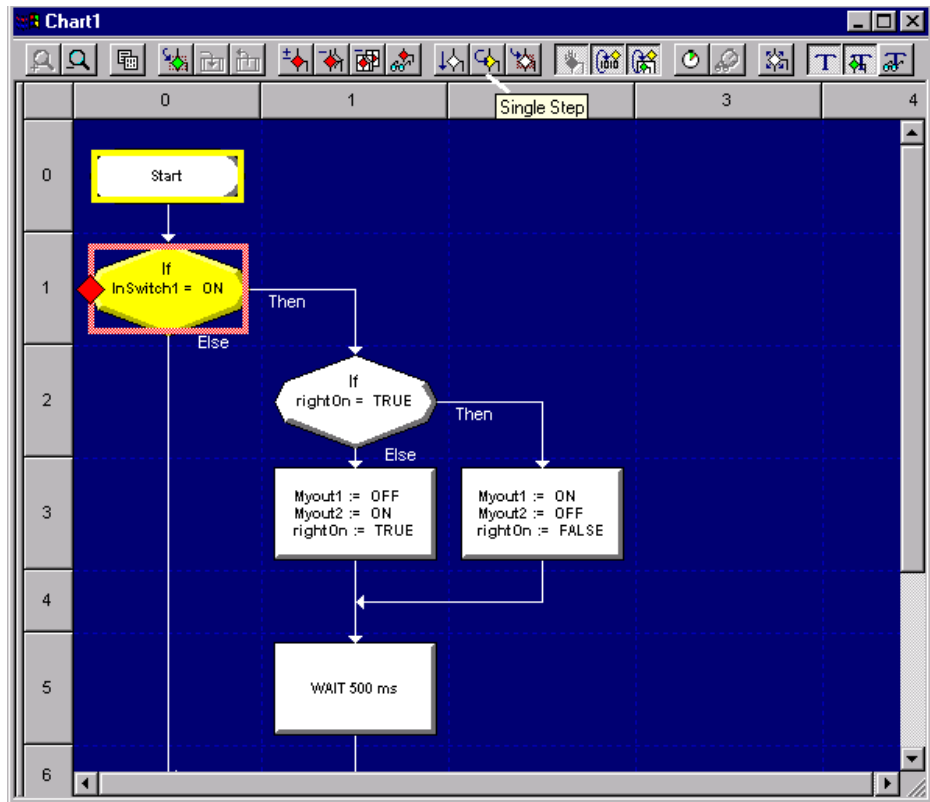


In a project as small as this (a single small flow chart), the executing program should hit the break point almost immediately. Note the pink rectangle around the block and the red diamond on the left side of the block. The pink rectangle

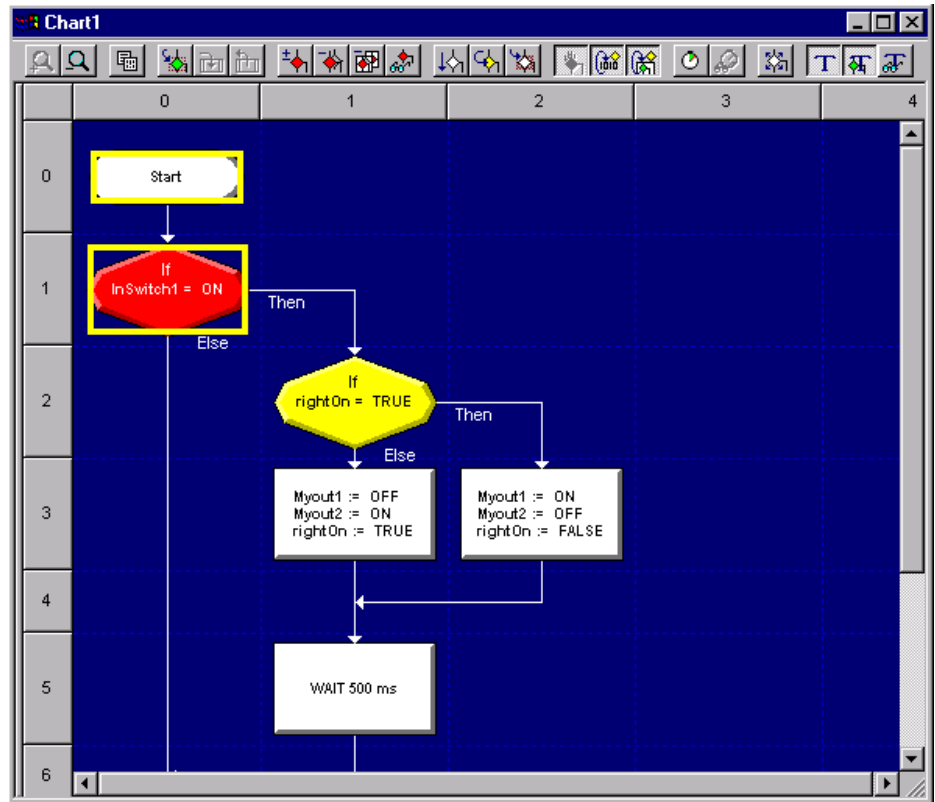
indicates that a break point is set for this block. The red diamond indicates that the program has hit the break point.

Now that we have hit a break point, we can step through the program one block at a time:

- 5. Click the **Single Step** tool on the toolbar.



- Continue to click the Single Step tool and watch the Pointe Controller step through your program.



Each time you click the Single Step tool, the program will step to and highlight the next block in the flow. All blocks will execute as they normally would, updating tag values and changing outputs as they go.

NOTE: You may wish to disable Debug Trace as you step through the program, in order to reduce screen clutter.

The simple program that you have just written, downloaded and monitored has given you a firm basis for any future program development. We have gone through all of the basic operations. You should be ready to develop and implement more complex "real" applications by following the same procedures.

As you begin development of your target application programs, recognize that in most such applications, it makes sense to use multiple flow charts, ladder diagrams, or a combination of both. You can experiment with the process on your own, or you can read the rest of the manual to get more detailed information.

Chapter 4: System Design and Installation

Now that you understand the basics of the Pointe Controller, you can design a complete machine control system using it. Good system design includes defining your I/O needs, selecting the appropriate modules and panels, calculating the power budget of the system, wiring the I/O to the equipment, and observing proper safety guidelines.

4.1 Safety Guidelines

WARNING: Providing a safe operating environment for personnel and equipment is your responsibility and should be your primary goal during system planning and installation. Automation systems can fail and may result in situations that can cause serious injury to personnel or damage to equipment. Do not rely on the automation system alone to provide a safe operating environment. You should use external electromechanical devices, such as relays or limit switches, that are independent of the Pointe Controller program to provide protection for any part of the system that may cause personal injury or damage.

Every automation application is different, so there may be special requirements for your particular application. Make sure you follow all national, state, and local government requirements for the proper installation and use of your equipment.

Plan for Safety

The best way to provide a safe operating environment is to make personnel and equipment safety part of the planning process. You should examine every aspect of the system to determine which areas are critical to operator or machine safety. If you are not familiar with Pointe Controller system installation practices, or your company does not have established installation guidelines, you should obtain additional information from the following sources.

- NEMA — The National Electrical Manufacturers Association, located in Washington, D.C., publishes many different documents that discuss standards for industrial control systems. You can order these publications directly from NEMA. Some of these include:
 - ICS 1, General Standards for Industrial Control and Systems
 - ICS 3, Industrial Systems
 - ICS 6, Enclosures for Industrial Control Systems
- NEC — The National Electrical Code provides regulations concerning the installation and use of various types of electrical equipment. Copies of the NEC Handbook can often be obtained from your local electrical equipment distributor or your local library.
- Local and State Agencies — many local governments and state governments have additional requirements above and beyond those described in the NEC Handbook. Check with your local Electrical Inspector or Fire Marshall office for information.

Safety Techniques

The publications mentioned provide many ideas and requirements for system safety. At a minimum, you should follow these regulations. Using the techniques listed below will further help reduce the risk of safety problems.

- Orderly system shutdown sequence in the Pointe Controller program.
- Emergency stop switch for disconnecting system power.

Orderly System Shutdown

The first level of protection can be provided with the Pointe Controller program by identifying machine problems. Analyze your application and identify any shutdown sequences that must be performed. Typical problems are jammed or missing parts, empty bins, etc. that do not pose a risk of personal injury or equipment damage.

WARNING: The control program must not be the only form of protection for any problems that may result in a risk of personal injury or equipment damage.

System Power Disconnect

By using electromechanical devices, such as master control relays and/or limit switches, you can prevent accidental equipment startup. When installed properly, these devices will prevent any machine operations from occurring.

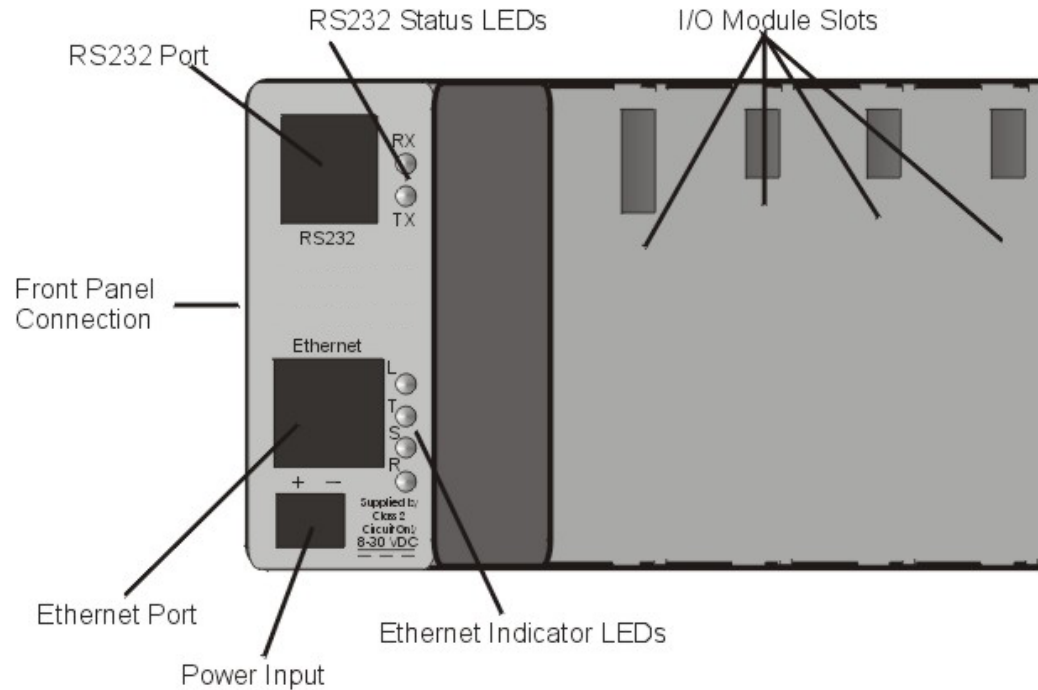
For example, if the machine has a jammed part, the Pointe Controller program can turn off the saw blade and retract the arbor. However, since the operator must open the guard to remove the part, you must include a bypass switch to disconnect all system power any time the guard is opened.

The operator must also have a quick method of manually disconnecting all system power. This is accomplished with a mechanical device clearly labeled as an Emergency Stop switch.

After an emergency shutdown or any other type of power interruption, there may be requirements that must be met before the Pointe Controller program can be restarted. For example, there may be specific register values that must be established (or maintained from the state prior to the shutdown) before operations can resume. In this case, you may want to use retentive memory locations, or include constants in the control program to ensure a known starting point.

4.2 Getting to Know the Pointe Controller Base

The figure below shows the layout of a Pointe Controller base:



The Pointe Controller base consists of a card cage containing the motherboard. The base unit has a built in Ethernet port, as well as an RS232 port. The Ethernet port is the interface to the larger system. The RS232 port is provided for general purpose communications (as defined by your application program). It is also designed to allow you to load future program upgrades (to incorporate the ability to interface future I/O boards and operator panels) into the base.

Both communications ports have status indicator LEDs which provide you with visible indications of each port's operation. The RS232 serial port indicates when it is transmitting (TX) and receiving (RX). The Ethernet port provides indications for good Ethernet link connection (L) and Ethernet port access by the base processor (S), as well as transmit (T) and receive (R) indicators.

Power must be provided to the unit by an external DC power supply. Any DC voltage within the range of 8-30VDC is acceptable.

Input and output modules can be plugged into the slots in the base. Most modules can plug into any base slot (including slot 0).

NOTE: Slot 0 includes additional features used by certain 12-pin specialty modules. These modules are documented as slot 0-specific.

The OptiLogic base can snap onto any standard DIN rail, including the rail molded into the back of all OptiLogic operator panels. When attaching an OptiLogic base to an OptiLogic operator panel, the 10-pin cable connection on the side of the base is used.

4.2.1 PTC-5800 Pointe Controller Technical Description

Physical (Base Unit)

- DIN rail mount to 35mm DIN rail
- Overall dimensions: 8.4"L x 3.25"H x 3.00" D
- Color: Dark gray
- Material: Polycarbonate plastic
- # I/O slots: 8

Environmental

- Storage Temperature : -20 to 70 C
- Ambient Operating Temperature : 0 to 55C
- Humidity : 0 - 95% non-condensing

Electrical

- Power: 8 - 30 VDC input power
- Minimum load current (no I/O boards or operator panel attached):
 - 75mA @ 24VDC
 - 150mA @ 12VDC
- Maximum load current (actual depends on the particular modules attached):
 - 700 mA @ 24VDC
 - 1.4A @ 12VDC
- Power available to I/O modules : 2.8A @ 5VDC
- Power Connection: Terminal block, 2 terminal

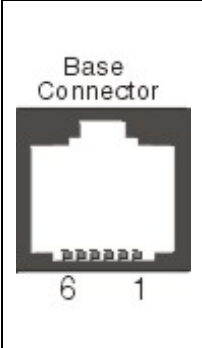
Communications, Ethernet

- Type: 10Base-T Ethernet
- Data Rate: 10 Mbps
- Connection: RJ45
- Ethernet Protocols: TCP/IP, OptiLogic UDP/IP, Modbus

TIP: For a complete description of Ethernet connections, see page 103.

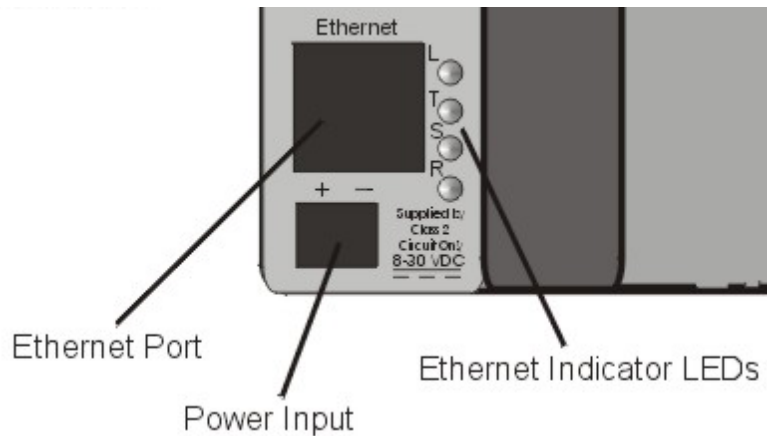
Communications, Serial

- Baud rates: 300, 1200, 2400, 4800, 9600, 19.2K (selectable)
- Data bits: 7 or 8 (selectable)
- Parity: odd, even, or none (selectable)
- Stop bits: 1 or 2 (selectable)
- Connection: RJ12

	Pin #	Description
	6	GND
	5	5V power out
	4	Transmit data (TX)
	3	Receive data (RX)
	2	Reserved (do not connect)
	1	GND

4.3 Supplying Power to the Controller

The Pointe Controller unit requires a 8-30VDC, 1 Amp power supply. This can be provided either by using a wall-pluggable AC adapter (part number OL-PS1) or by connecting the unit directly to a properly rated DC grid. The connection is made at the Power Input screw terminals located at the bottom left corner of the base unit.



To connect a power supply to the controller:

1. Make sure the power is OFF – the AC adapter should be unplugged and/or the DC grid should be turned off.
2. Using a regular slotted screwdriver, loosen the Power Input screw terminals.
3. Pass the power supply wires through the opening in the bottom of the controller base unit. Insert the positive wire into the positive terminal (+) and the negative wire into the negative terminal (-).
4. Retighten the screw terminals.
5. Tug gently on the wires to verify that they are properly secured to the terminals.

The Pointe Controller unit can now be powered on.

4.4 Installing the PointeControl Software

The PointeControl software CD (part number NS-PTC) includes the control application development package and the various utilities needed to connect to and configure the Pointe Controller unit.

System requirements:

- 200 MHz or faster Pentium processor
- Operating system (any one):
 - Microsoft Windows NT 4.0 Service Pack 5 or 6
 - Microsoft Windows 2000
 - Microsoft Windows XP
- A CD-ROM drive
- A 10BaseT Ethernet card
- A DB-9 serial port

To install the PointeControl software on your PC:

1. Insert the PointeControl software CD into your CD-ROM drive.
2. Open the mounted CD (typically drive D: or E:) and double-click SETUP.EXE.
3. Follow the onscreen installation instructions. No unusual installation options are presented.
4. Restart your PC when prompted.

NOTE: As part of the PointeControl software installation, a Java Runtime Engine (JRE) is also installed on your PC. This JRE is used only by the PointeControl software and it is not included in the Windows registry. It should not conflict with any other Java tools you may have installed on your PC.

4.5 Addressing the Pointe Controller

Each Pointe Controller unit has two distinct addresses: an IP address, for communicating across an Ethernet network; and a Modbus address, for communicating with serial Modbus devices such as operator panels and bar code readers.

4.5.1 IP Address

The Pointe Controller unit comes preconfigured with a default IP address. You must reset the address so that the unit can properly communicate on your Ethernet network. This change is made via a direct serial connection between your PC and the Pointe Controller unit.

Remember that each Pointe Controller unit on your network must have its own unique IP address and node name, which is set prior to applying power to the controller. Duplicate addresses will cause system communications to fail.

To set the IP address of the Pointe Controller unit:

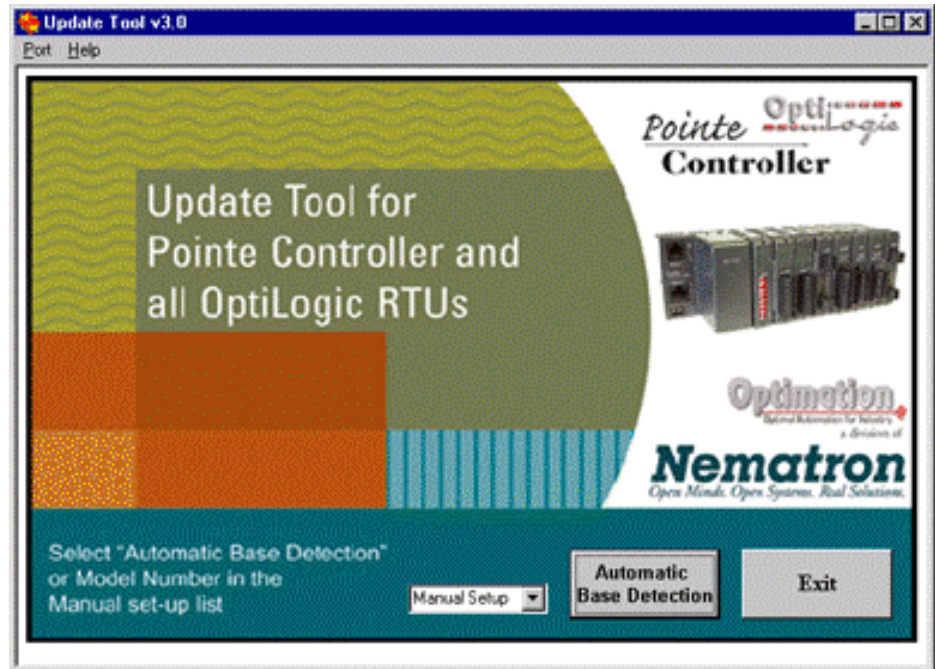
1. Establish a serial connection between your PC and the Pointe Controller unit, using the download cable (OL-CBL-DNL):
 - a. Before you connect the serial cable, make sure the Pointe Controller is powered off. The unit looks for the cable when it is first powered on.
 - b. Connect the cable's RJ-11 plug to the Pointe Controller's serial port.
 - c. Connect the cable's DB-9 plug to your PC's serial port.
2. Power on the Pointe Controller unit.

NOTE: There is no power switch on the Pointe Controller unit itself. Either the AC adapted must be plugged in or the directly connected DC grid must be turned on.

3. From the Windows **Start** menu, choose **Programs > PointeControl > Update Tool**.

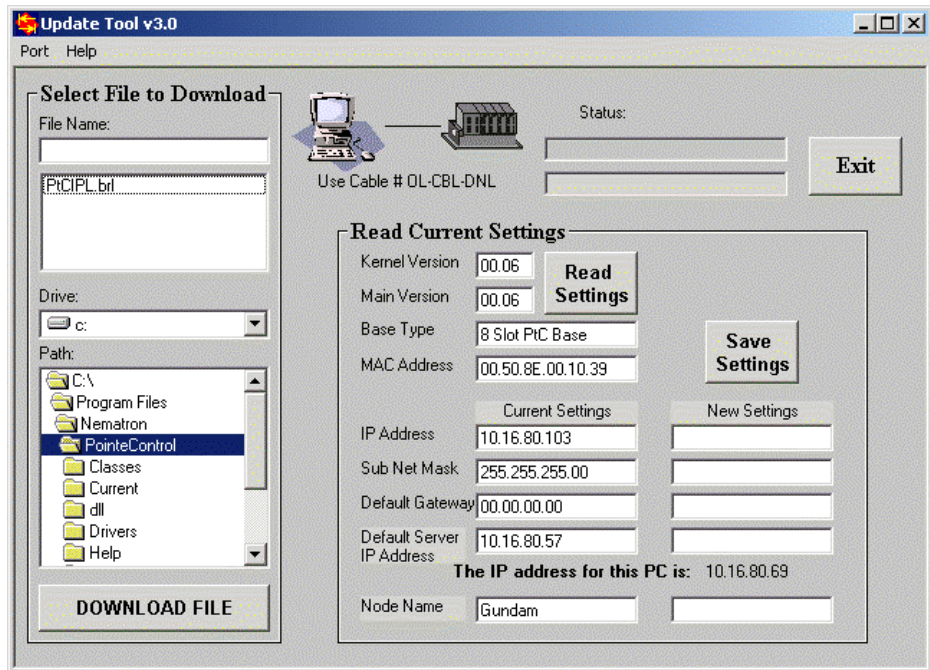
NOTE: If you are running the Update Tool for the first time, you will be asked to specify which COM (serial) port the tool should use. Enter the number (1, 2, 3, or 4) to which you connected the serial cable in Step 1 above, and then click **OK**. After that, the tool will finish launching.

The PointeControl/OptiLogic Update Tool application window appears.



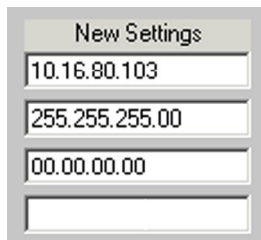
4. Click **Automatic Base Detection**. The application should immediately connect to your Pointe Controller unit. If it does not, check your serial connection and try again. If it still cannot connect, click **Manual Setup** and select **PTC5800** from the drop-down menu.

When the application successfully connects, the following window will appear:



The **Read Current Settings** pane displays the current address settings on the Pointe Controller unit. If you are addressing the unit for the first time, the factory default settings are displayed.

5. Under **New Settings**, enter the new IP address and subnet mask for the Pointe Controller unit. For example, an IP Address of "10.16.80.103" and a Sub Net Mask of "255.255.255.00".



The Pointe Controller unit should receive an address on the same subnet as your PC. If you do not know what values to enter, contact your system administrator.

NOTE: The Pointe Controller unit does not communicate directly with any network gateway or router. Instead, it broadcasts to all machines on its subnet. Therefore, you should enter "00.00.00.00" in the Default Gateway field.

- For the **Default Server IP Address**, enter the IP address of the PC with which you are connecting to the Pointe Controller unit. For example, "10.16.80.69".

The screenshot shows a window titled "New Settings" with four input fields. The first field contains "10.16.80.103", the second "255.255.255.00", the third "00.00.00.00", and the fourth "10.16.80.69".

The secondary server is the PC to which the Pointe Controller unit will attempt to connect when it first powers on.

- For the **Node Name** pane, enter the name by which the Pointe Controller unit will identify itself to PointeControl Monitor. For example, "Gundam."

NOTE: If you do not want or need to change the Node Name, you can skip this step and leave the factory default setting.

For more information on PointeControl Monitor, see Chapter 6, "Downloading to the Controller," and Chapter 7, "Monitoring and Debugging."

- Click the **Save Settings** button to save your settings to the Pointe Controller unit. When the settings are saved, the fields will turn green.

The screenshot shows the "Save Settings" button at the top. Below it is the "New Settings" section with four input fields, all of which are highlighted in green. The values in the fields are "10.16.80.103", "255.255.255.00", "00.00.00.00", and "10.16.80.69". Below these fields, the text "PC is: 10.16.80.69" is displayed. At the bottom, there is another green-highlighted input field containing the text "Gundam".

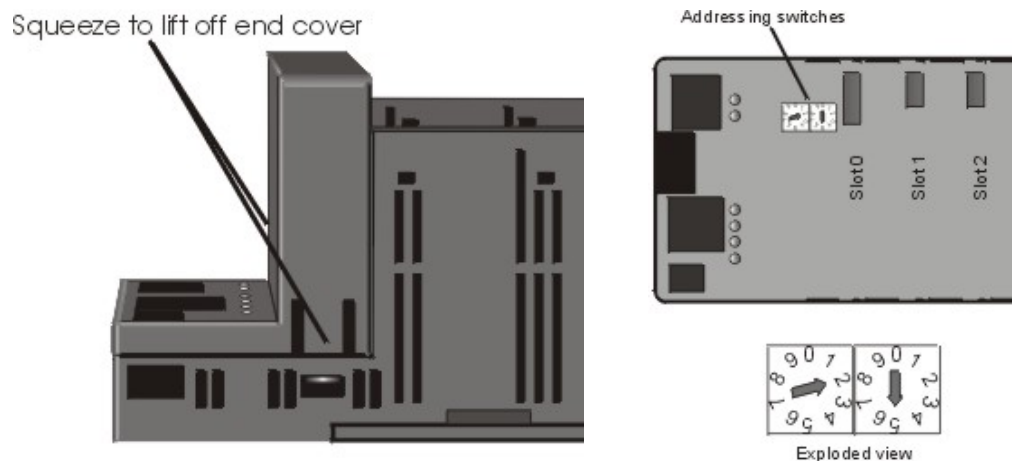
- Click the **Read Current Settings** button to verify that the new settings were saved correctly. The current settings should now match the IP address, subnet mask, secondary server, and node name that you entered.
- Exit the Update Tool application by clicking the **Exit** button.
- Power off the Pointe Controller unit.
- Disconnect the RS-232 serial cable.

4.5.2 Modbus Address

The addressing that you, the system designer, must set is the address set via rotary address switches in the Pointe Controller base unit. Each controller in your system must have its own unique address. This address, a value between 00 and 97, is how the software in the master PC identifies each controller.

NOTE: Addresses 98 and 99 are reserved for performing hardware resets. For more information, see page 230.

To get to the address switches, you must first remove the end cover from the base unit. To do this, simply squeeze the latching tabs, shown in the figure below, and lift the cover off.



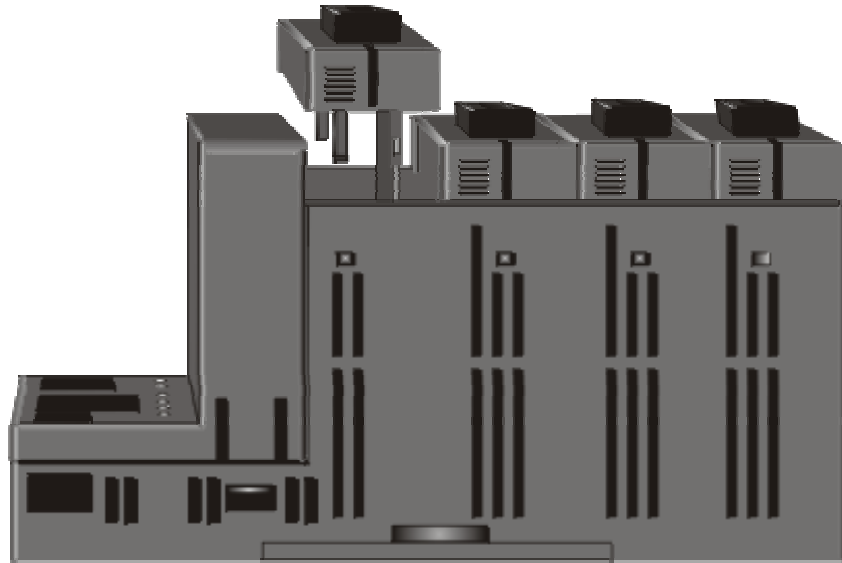
Removing the end cover will expose the base motherboard. The address switches will be found near the connector for slot 0.

To set the controller's Modbus address, rotate the switches to the desired values. The switch on the left is the "tens" digit. The switch on the right is the "ones" digit. A small flat blade screwdriver is the only tool you need. The address shown on the figure above is "25."

Remember that each Pointe Controller unit on your network must have its own unique Modbus address, which is set prior to applying power to the controller. Duplicate addresses will cause system communications to fail.

For more information on configuring Modbus communications, see Chapter 8, "Networked Operations," starting on page 216.

4.6 An Overview of OptiLogic I/O



The Pointe Controller system achieves its modularity and flexibility by integrating with Optimization, Inc.'s complete line of OptiLogic I/O modules and operator panels. (Optimization is a wholly-owned subsidiary of Nematron.) OptiLogic components can be plugged together in nearly any combination. This chapter covers the currently available modules that plug into the card cage.

TIP: Additional I/O modules are always under development. Please check our Web site at <http://www.nematron.com/PointeControl> for a complete list of available modules.

Most OptiLogic modules can be installed in any card cage slot and used in any combination and quantity that will fit in the card cage. This applies to all general purpose digital and analog I/O. If you need all digital inputs — plug in digital input modules only. If you need a mixture of analog and digital inputs and output — select the mixture that fits your needs. Snap together modularity gives you the ability to optimize your system for your needs.

OptiLogic I/O modules are designed to meet your needs in real world application. They are all small circuit boards with a few available points to minimize your system cost. Most module connectors are pluggable terminal strips for easy connection, and easy maintainability. The snap-together design means low labor costs — or costs on your time. Visual status indicators on digital I/O and communications modules provide a convenient means for monitoring operation. All together, the result is a cost effective, easy to use and maintain set of industrial control hardware.

This manual covers general I/O characteristics and applications first. Specific I/O boards are covered in the latter pages. The general pages should serve as a guide to selecting and installing I/O boards in your application.

4.6.1 Digital Inputs

Digital I/O modules are used to either monitor (input) or control (output) the “state” of something. “State” being on or off, active or inactive, open or closed — etc. In the “real world” digital I/O requirements come in a variety of shapes and sizes. Therefore, there are a variety of available modules designed to meet the variety of needs.

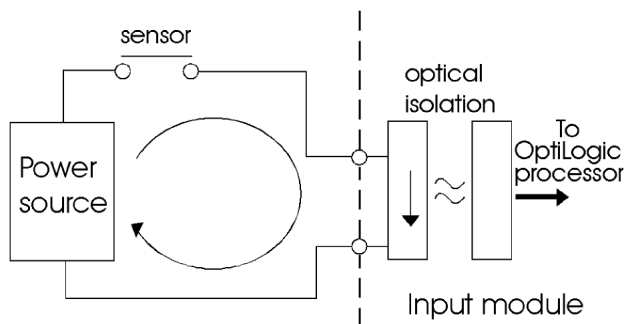
Typical digital inputs are connected to switches, buttons, digital outputs from other equipment, discrete level sensors, thermostats and other on/off sensing devices.

Digital status is sensed by a controller, such as an OptiLogic system, by passing current through an input sensor. When the current is on, the input state is active. When it is not there, the input state is inactive.

Input Isolation

In most cases, it is important to “isolate” the real world inputs from the internal electronics of the controller. You want to prevent some external situation from “zapping” the controller’s electronics.

An effective means of providing such electrical isolation is *optical isolation*. The figure below illustrates the basic concepts of optical isolation of a digital input circuit:

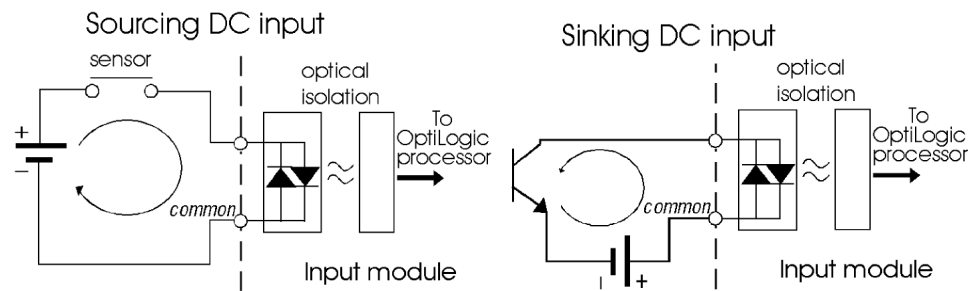


As shown, when the digital input contact closes, the circuit path is complete and current will flow. On the input module this circuit path passes through a device which emits light when current flows through it. The light emitter is in very close physical proximity to (actually in the same chip) a photo sensor, which will turn on when it senses light. In this way, a digital input module can sense whether the input device is closed (current flow) or open (no current flow) without a direct electrical connection between the external sensor and the internal electronics.

DC Inputs

DC digital inputs are typically supplied by a DC power supply. The most common DC supplies used in industry are 12VDC and 24VDC.

Typical DC digital input circuits are shown below:

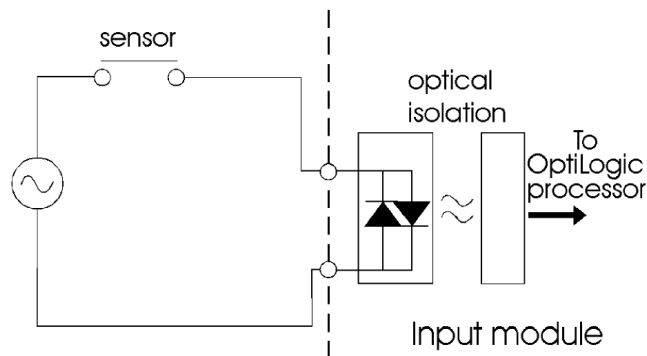


As shown, the physical optical emitter on the input module is an LED (light emitting diode). OptiLogic DC inputs use bidirectional LEDs — i.e. Your inputs may either source or sink current. The top figure shows a sourcing input. The figure below it shows a sinking input. When inputs are connected to a “common” (most instances), inputs must be either all sourcing or all sinking.

AC Inputs

AC digital inputs are typically supplied either directly from line voltage or transformed down from line voltage. The most common AC inputs are 120VAC and 24VAC, although any voltage range is possible.

A typical AC input circuit is shown below:



As shown, the physical optical emitter on the input module consists of two LEDs of opposite polarity. An AC (alternating current) connection flows current one way, then the other. Light is emitted in both cases.

There is a short period when voltage, and therefore current flow, switches from one direction to the other when no current flows. This is called zero crossover. During zero crossover, the digital input circuit must “debounce” the signal to ensure that the system does not provide a false indication that the input contact is not closed when it is, in fact, closed. OptiLogic AC digital inputs handle such zero crossover conditions.

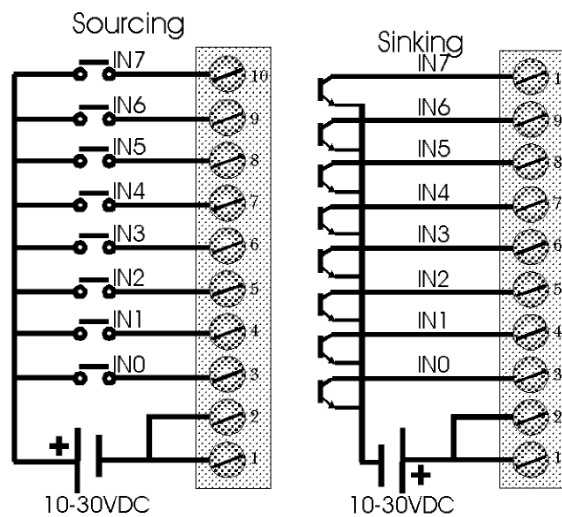
Digital Input Voltage

Any digital input module, AC or DC, is designed to operate within an input voltage range. The input voltage directly controls the amount of current flowing through the circuit. The minimum voltage corresponds to a voltage that creates enough current to produce LED light sufficient to be sensed by the optical sensor. The maximum voltage corresponds to the maximum current the optocoupler can handle without being damaged.

I/O "Common" Terminals

For a digital input circuit, one input terminal and one output terminal is necessary for operation. For practical application, one of these two terminals may be "common" to several circuits.

In most systems, the power source for all digital inputs is from the same supply. In such cases, connecting all of the circuit return lines together results in reduced equipment costs as well as simpler system wiring.



The example above illustrates a digital input board that has eight inputs and two commons. This can be accomplished with a 10 terminal connector block.

4.6.2 Digital Outputs

Digital outputs are used to turn “loads” on and off. “Loads” may be lights, motors, solenoids, or any type of on/off device found in the “real world.”

Digital outputs in the OptiLogic series come in three types — relay, transistor and solid state relay. Each type has applications it is best suited for. The following is a general list of application characteristics for each output type:

Relay	Transistor	Solid State Relay
<ul style="list-style-type: none"> ▪ Low contact loss ▪ AC or DC ▪ Moderate to high current rating ▪ Low cost ▪ Should not be used for: <ul style="list-style-type: none"> ○ Ultra low current switching (less than 10mA) ○ Switching loads at high frequency 	<ul style="list-style-type: none"> ▪ DC application only ▪ Low current rating ▪ High frequency switching ▪ Low cost 	<ul style="list-style-type: none"> ▪ AC application ▪ Moderate current ▪ Any switching frequency ▪ Moderate cost

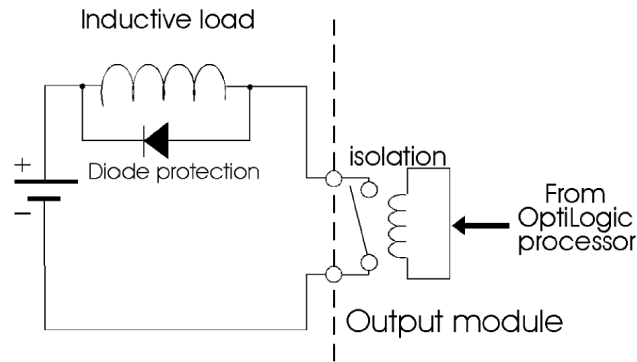
Relay Outputs

Relays are basically electrically controlled mechanical switches. All current OptiLogic Relay output boards utilize form A relays — i.e. the contact is either open or closed.

Relays are affected by the type of load that is switched. Inductive loads (solenoids, motors, etc.) tend to wear the relay much more than resistive loads (lights, heaters, etc.).

Inductive load wear is due to the fact that inductive loads will continue to conduct current for a period, even after the circuit is broken. This current flow builds up opposing polarity charges between the contact segments that just separated. This makes the two segments attract each other — making opening the contact more difficult. It also can result in arcing while the contact is being opened. Arcing, in turn, builds up carbon deposits, i.e. wear.

This situation can be improved for DC inductive circuit loads by the addition of external diode protection of the circuit. The figure below illustrates diode protection:

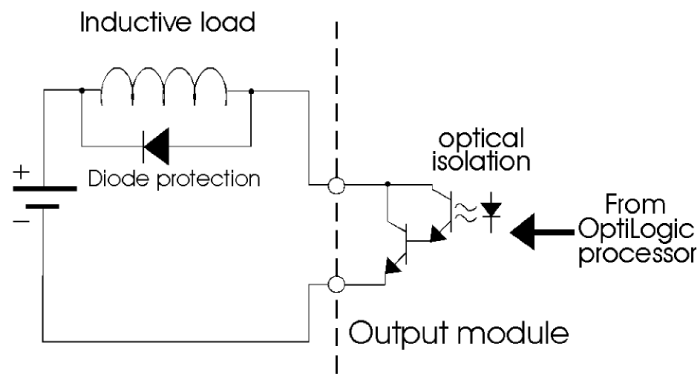


When the contact is closed, the diode is reverse biased and no current flows through it. When the contact opens, current will continue to flow through the inductive load. The diode provides a path for current flow. The result that is the energy is dissipated in the inductive coil and not the relay contact.

NOTE: Do not use this circuit for AC loads.

Transistor Outputs

An NPN transistor sinking output provides a path to ground. A typical circuit is shown below:



There is a small voltage drop across the transistor in such a circuit. The voltage drop will generate heat in the transistor. Therefore NPN transistor outputs are generally limited to lower current applications.

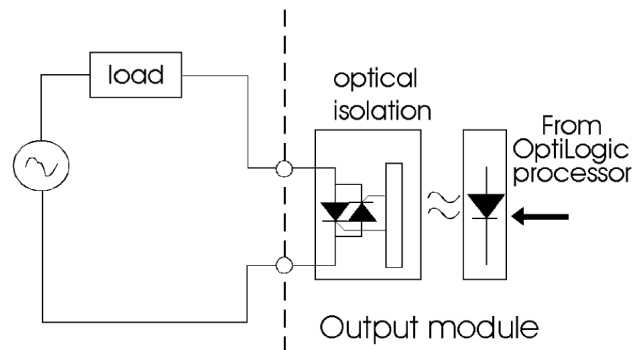
Transistor outputs can be operated at high frequency. There is no effective wear on a transistor output from switching, as there is in a mechanical relay.

Diode protection applied to inductive loads is recommended in cases where the load current approaches the rated current limit of the output. In most cases OptiLogic outputs are designed to withstand voltages of at least twice the rated output voltage. However, diode protection like that shown above will ensure that turn off voltage spikes will never get to that level.

Solid State Relay Outputs

Solid state relays are semiconductor switches that operate very much like mechanical relays. They have an advantage over mechanical relays by virtue of the fact that they are semiconductors. Solid state relays can be switched at relatively high frequencies and they do not wear out. However they are more expensive and there is a small voltage drop across the contact.

The figure below illustrates a typical solid state relay output:



OptiLogic Solid state relays are designed for AC load operation.

4.6.3 Analog Inputs

Analog inputs are used to monitor the value of some continuously variable measurement. Typical analog inputs are measurements of temperature, pressure, weight, liquid level, pH, flow rate and many other “real world” parameters.

The purpose of an analog input module is to convert the measurement into a format that is usable by the data acquisition or control system. To be usable by a computer-based system, the analog measurement must be converted to digital format. Doing so accurately and, in some cases, quickly, is the goal of the analog to digital converter module.

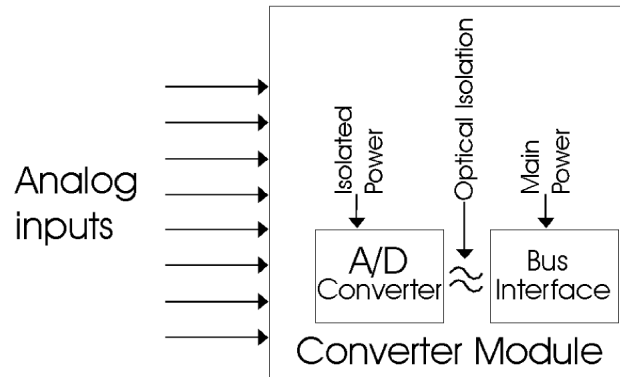
A good understanding of analog input modules includes an understanding of isolation, accuracy, single and differential inputs, multiplexing, resolution and range. The following paragraphs provide an overview of these subjects.

Isolation

In many applications there is a good deal of benefit to be derived from isolating the analog measurement source from the RTU’s power supply. In some cases, signal inputs may contain voltages or noise signals which could adversely affect the main processor’s operation. Likewise, noise on the main power bus can degrade the accuracy of the analog value measurement. Both potential problems can be solved by isolating the analog inputs from the main power supply.

Isolation involves totally isolating the analog to digital (A/D) converter from the main power bus. This can be accomplished in two ways. The A/D input module can use a separate power source input, which is isolated from the power input to the base. The A/D module can also use the main power supply and isolated power

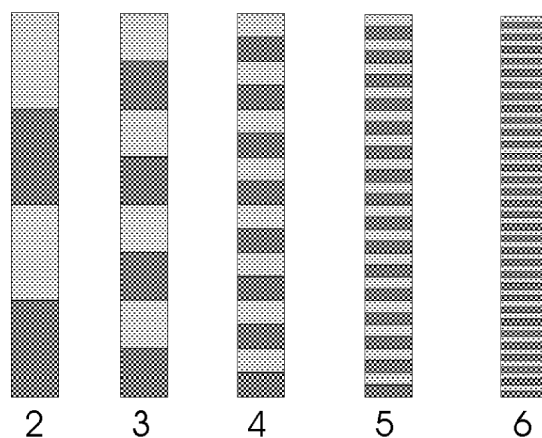
via a switching power converter and a transformer. There are OptiLogic analog input modules in both categories. Neither is functionally superior to the other. The on-board power generation may save the cost of an additional external power supply.



The other aspect of isolation is the fact that the measured value must be transmitted from the analog to digital converter, operating on one power supply, to the main system, which is operating on another power supply. This is commonly accomplished through optical isolators.

Resolution

Resolution is the number of significant bits of information the A/D converter uses to express the value of the measured input. A 12-bit A/D converter uses 12 bits of information, meaning the entire range is covered by a number between 0 and $(2(12)-1)$ or 0 to 4095. A 14-bit A/D expresses the same range as a number between 0 and 16,383. In other words, the more bits used, the finer the increment. In general terms, the higher the resolution, the better.



Accuracy

Accuracy is expressed as the worst case deviation from the “ideal value” across the entire input range. For example, for a 0 to 5V input range and a 12-bit A/D module, a 2.0 volt input should yield a value equal to 1638 (0.4 x 4096). If it returns a value of 1636, and this is the worst case error across the entire range of 0 to 5V, the accuracy is 12 bits +/- 2 counts.

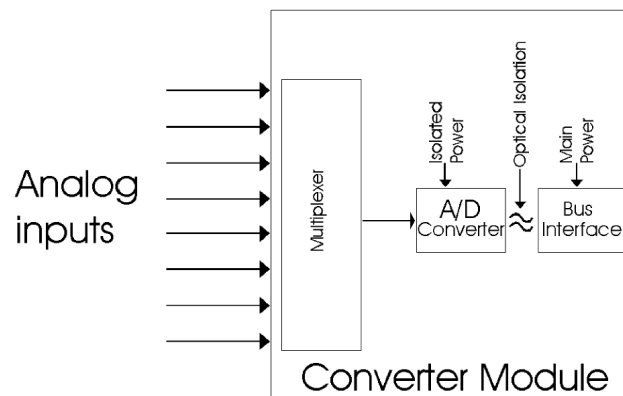
Range

The analog input range is the minimum and maximum voltage, or current level, measured by the A/C converter. Typical ranges are 0 to 5 volts, 0 to 10 volts, +/- 5 volts, +/- 10 volts, and 4 to 20 mA.

You should try to match the input range to the range of the signal that you are measuring.

Multiplexing

Analog to digital converter devices are typically quite expensive. In order to keep the cost per channel of analog inputs down, a multiplexer is commonly used.

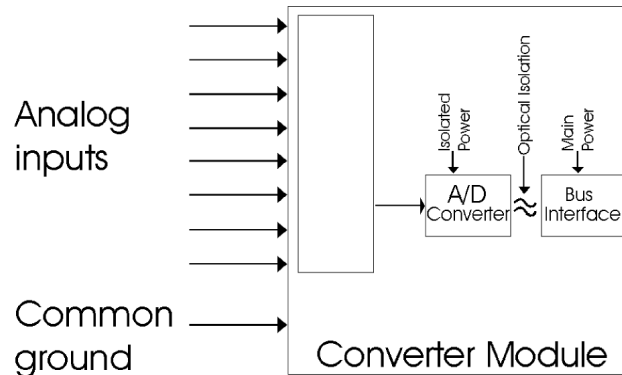


A multiplexer switches one analog input at a time into the A/D converter. Each input is converted in sequence. The trade off is reduced sampling rate for a particular channel versus reduced cost per channel measured. In most industrial applications, the conversion rate is so fast in relation to the rate of change in the measured value, that sampling rate is not a factor.

Single Ended Inputs

Single ended inputs are all referenced to the same ground point. In many applications, single ended inputs produce significant advantages. Single ended inputs require only one ground connection and one signal input per measured value. The result is reduced wiring costs along with the reduced cost per channel on the analog input module.

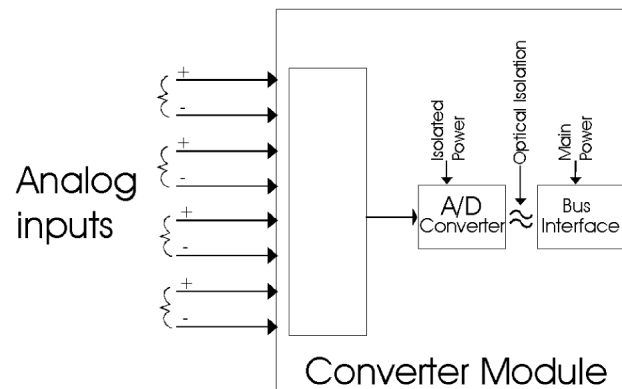
In order to use single ended inputs, the ground connection must be very good. The measurement devices must also be capable of being referenced to a common ground.



Differential Inputs

There are cases when the individual analog inputs cannot be connected to a common ground. In those cases, a differential input A/D converter should be used.

With a differential analog input, both a positive and negative signal line must be connected for each signal. The analog input module then measures the difference between the positive and negative. The effect of one channel's signal on another channel's signal should be as little as possible. That relationship of the effect on the measured value of one channel to the value input on a second channel is called "common mode." The higher the "common mode rejection ratio (CMRR)" the better.



4.7 Determining Your I/O Needs

The Pointe Controller system offers the following ways to add I/O to the system:

- Local I/O – consists of I/O modules installed in the Pointe Controller unit itself.
- Remote I/O – consists of I/O modules installed in OptiLogic RTU bases connected to the Pointe Controller unit through the Ethernet network.

A Pointe Controller system can be developed using many different arrangements of these configurations. All I/O configurations use the standard complement of OptiLogic I/O modules and bases.

TIP: For complete technical descriptions of all OptiLogic modules and panels, see Appendix A, "OptiLogic Technical Specifications," starting on page 231.

4.7.1 Available I/O Modules

The following is a list of I/O and operator panel modules available at the time of this printing. Keep in mind that the range of available OptiLogic modules is always expanding. Many more modules will be available in the near future. To get a current list of available modules, visit our Web site at <http://www.nematron.com/PointeControl>.

The following I/O modules are currently available (generally off the shelf):

Digital Inputs	
OL2201	8 Digital input simulator (toggle switch input)
OL2205	4 AC/DC (10-30V) In -- Each input has a separate common
OL2208	8 DC (10-30VDC) In
OL2211	8 AC (80-132VAC) In
OL2252	2 high speed counters (up to 20KHz) inputs. 6 additional inputs configurable as general purpose DC inputs or control signals.
OL2258	High speed counter input for pulse encoder type devices. Up/Down count, Pulse & Direction or Quadrature inputs accepted. Pulse counting to 80KHz (160KHz for quadrature). Two high speed transistor outputs.
Digital Outputs	
OL2104	4 Relay (2A resistive @ 24VDC, 1A @120VAC)
OL2108	8 Relay (2A resistive @ 24VDC, 1A @120VAC)
OL2109	8 Transistor (500mA sink)
OL2111	8 AC Solid State Relay (1A)
Analog Inputs / Outputs	

OL2304	4 channel voltage output, 0-5V, 0-10V, +/-5V, +/-10V
OL2408	8 channel 0-5VDC or 0-10VDC in
OL2418	8 channel 4-20mA in
Communications	
OL2602	2 Port RS232

4.7.2 Available Operator Panels

The following is a list of currently available OptiLogic Operator Panels:

Pushbutton / Indicator Panels	
OL3406	6 Indicator/4 Pushbutton Alphanumeric Display
OL3440	4 Line x 20 Character backlit LCD alphanumeric display
Terminal Panels	
OL3420	2 Line x 20 character backlit LCD display, 4 pushbuttons
OL3850	2 line x 20 character backlit LCD display, 5 user definable pushbuttons, numeric keypad, 3 indicator light bars

4.7.3 Calculating Your Power Budget

Each I/O module and operator panel that you install in your Pointe Controller unit requires a certain minimum amount of power to operate. Each controller base has a maximum of 2.8 A (2800 mA) total power available.

To calculate the power budget for your Pointe Controller, simply add up the power required for all of the I/O modules and operator panels that you want to install in the controller. If the total power required is less than or equal to 2.8 A, then the controller will be able to power everything. If the total power required is greater than 2.8 A, then you must redesign your application to use a different combination of modules.

The table below provides a quick reference to the power requirements for all of the available I/O modules and operator panels. Again, for complete technical descriptions of these components, see Appendix A, "OptiLogic Technical Specifications," starting on page 231.

Module Type	Description	Power Req.
OL2104	Relay Output, 4-point	250 mA
OL2108	Relay Output, 8-point	375 mA
OL2109	DC Sinking Output, 8-point	140 mA

Module Type	Description	Power Req.
OL2111	Solid State Relay Output, 8-point	120 mA
OL2201	Digit Input Simulator, 8-point	60 mA
OL2205	AC/DC Digital Input, 4-point	100 mA
OL2208	DC Digital Input, 8-point	60 mA
OL2211	AC Digital Input, 8-point	100 mA
OL2252	Dual Pulse Counter	100 mA
OL2258	High Speed Pulse Counter	400 mA
OL2304	Analog Voltage Output, 4-channel	700 mA
OL2408	Analog Voltage Input, 8-channel	700 mA
OL2418	Analog Current Input, 8-channel	700 mA
OL2602	Dual Serial Port	110 mA
Panel Type	Description	Power
OL3406	Pushbutton/Indicator Panel	50 mA
OL3420	Operator Terminal	115 mA
OL3440	4-line Display Panel	150 mA
OL3850	Keypad Terminal	525 mA

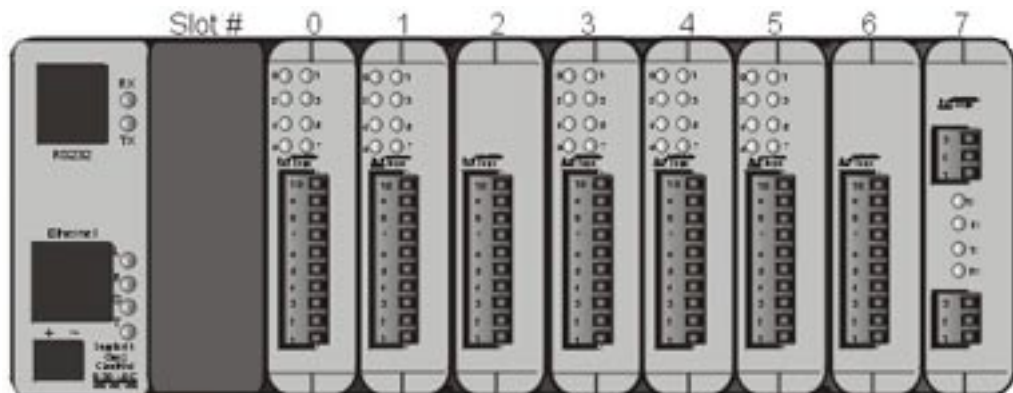
4.8 Installing I/O Modules in the Controller

Each Pointe Controller application will differ in the number and type of I/O modules, the operator panels used (if any) and how the devices must be distributed. The modular design of the Pointe Controller system allows you to mix and match to meet your exact application requirements.

System configuration entails an early process of defining exactly what type and quantity of I/O you need at each location. If operator interaction, alarm annunciation, or status display are required at the various points, the appropriate operator panel should be chosen. Once that is done, you can custom tailor your controllers by selecting and installing standard I/O modules in your Pointe Controller base units and snapping the appropriate operator panel onto each.

4.8.1 Slot Numbering

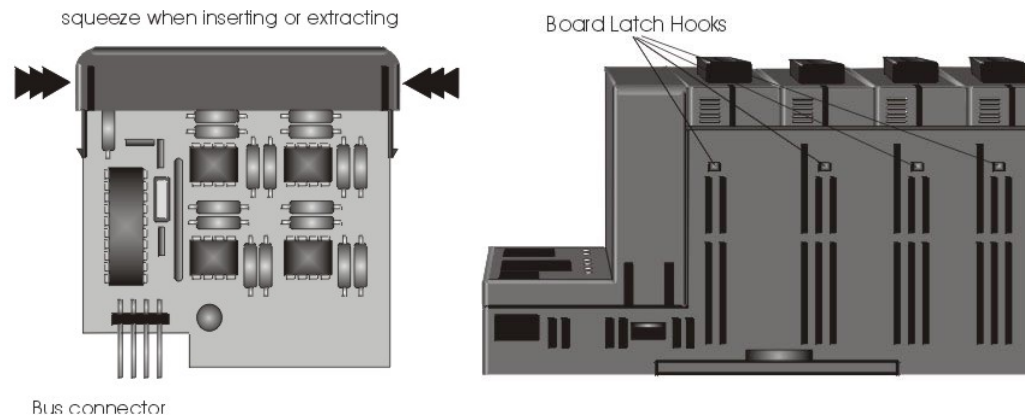
Each module will occupy one slot in the controller base. Each slot position is numbered as shown below. The slot number will provide a reference to your application program for selecting the appropriate module for each particular operation.



Slot numbering is simply left to right, starting with slot number 0.

4.8.2 Installing Modules

Each slot has card guides along each side and a connector on the motherboard. To install an I/O module, place the module's circuitry board in the top and bottom card guides. (Note that the board will not be tightly retained until it is approximately 3/4 inch into the card guide.)



As you push the module into its mating connector, squeeze the ends together. This will allow the board latches to travel inside the card cage. When you have pushed the board into its mating connector and released, the latches should hook the card cage and keep the module in place.

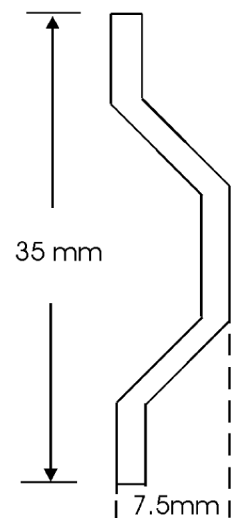
4.9 Mounting the Pointe Controller

Pointe Controller units are intended to be mounted on a standard DIN rail. That DIN rail can be a commercial DIN rail attached to any flat surface. It can also be the DIN rail built into OptiLogic operator panels.

4.9.1 Mounting the Base on a DIN Rail

A DIN rail is simply a standard “U” shaped channel which is designed to be mounted horizontally on any flat surface. DIN rail can be purchased at nearly any electrical supply outlet.

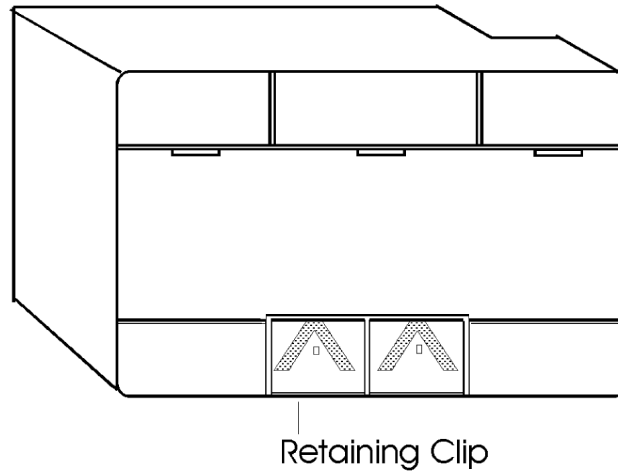
There are a few standard DIN rail sizes available. Pictured below is a cross sectional view of the standard 35 mm DIN rail the Pointe Controller base unit is designed to clamp on:



DIN Rail Dimensions

The key dimensions are the 35mm overall width and a minimum 7.5mm depth. The precise channel shape is not important.

Now, take a look at the bottom side of the controller base. It will appear as shown below:



The DIN rail channel runs lengthwise across the middle of the base's bottom side. At the top of that channel are three overhanging hooks. At the bottom of the channel there is a sliding retaining clip.

The process of installing a base on a DIN rail is as follows:

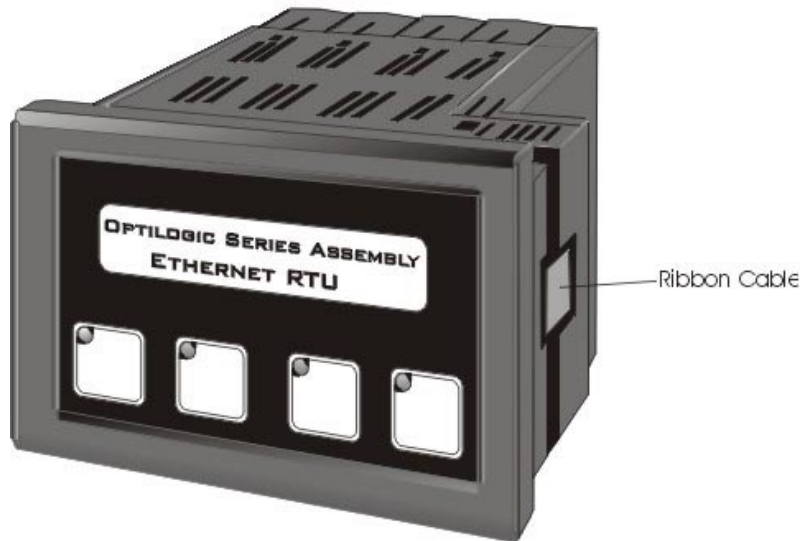
1. Pull the retaining clip back from the center of the base. It should pull back about 1/8 inch. The retaining clip on an uninstalled unit can be pulled back with your fingers.
2. Place the Pointe Controller base on the **horizontal** DIN rail with the three overhanging hooks over the top of the rail. Mounting must be horizontal to allow convection air flow for cooling.
3. Rock the Pointe Controller base down flat against the bottom of the DIN rail.
4. Push the retaining clip closed to hook the bottom rib of the DIN rail.

4.9.2 Mounting the Base to an Operator Panel

The Pointe Controller unit base can also be mounted to any OptiLogic operator panel. As shown in the figure below, OptiLogic operator panels have a built in DIN rail for mounting the base.



The mounting process is exactly the same as described for mounting to a DIN rail. Be sure that your orientation is right so the connectors on the base and the front panel line up. An Pointe Controller unit base attached to an OptiLogic operator panel should look like the figure below.



The short ribbon cable, which comes with the operator panel should be used to provide the connection between the Pointe Controller base and the operator panel.

4.10 Connecting the Controller to Your Network

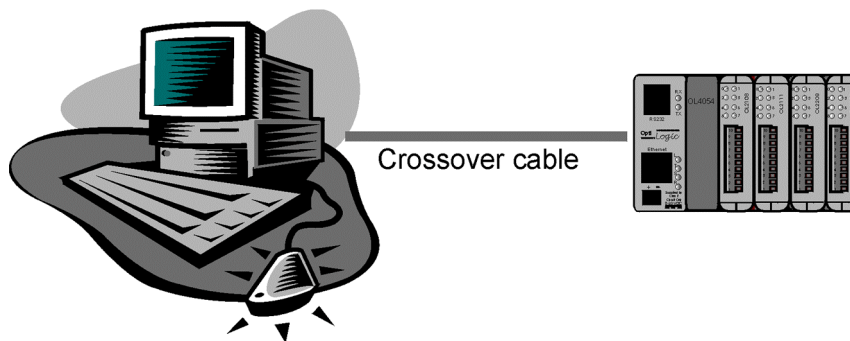
An Ethernet network connection is used to download finished control applications from your PC to the Pointe Controller unit. It is also used to monitor the unit's runtime performance and to share Modbus TCP data between different control devices. An RJ-45 Ethernet port is located at the left side of the Pointe Controller unit.

NOTE: The following sections discuss interconnection using the term hub. Any of the configurations apply equally well to a switch.

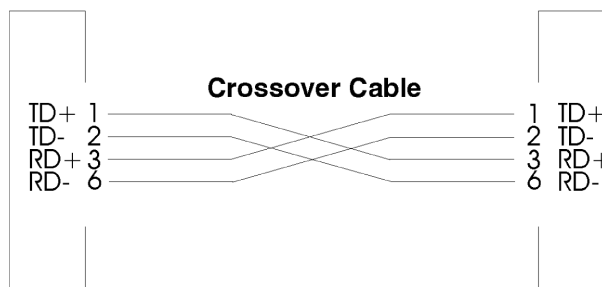
In an Pointe Controller system, there should be a single *master*, the host PC. All physical media interconnections should be made to commercial building wiring standards EIA/TIA-568 and the specification for Unshielded Twisted Pair cable defined in the TIA/EIA TSB40-A specifications. For best case 10Base-T wiring, we recommend using all CAT5 type cabling for connecting your Pointe Controller network.

4.10.1 Point-to-Point Connection

The simplest system is a point-to-point connection. Point to point connections, as illustrated below, require only a *crossover* type patch cable.

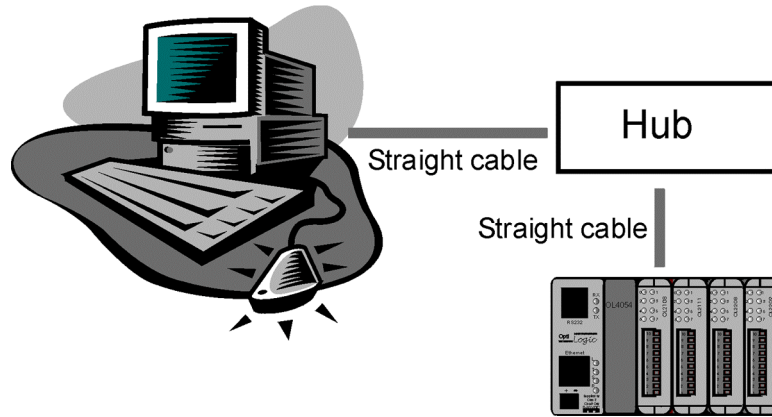


An Ethernet crossover cable, shown below, connects the transmitter on one side, with the receiver on the other. This is a category 5 type UTP crossover patch cable. Cable length is limited to less than 100 meters.



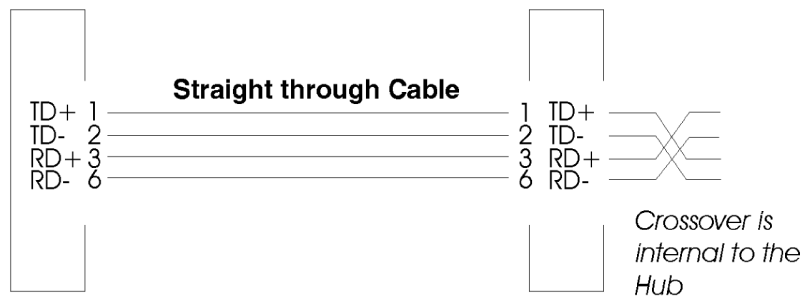
4.10.2 Single Hub and Switched Connections

The next level of complexity is a single hub or switch system. Hubs and switches are commonly available with anywhere from 4 to 24 connections.



The multiple Ethernet ports on a hub allow physical star type network wiring. The hub is typically placed in the center of the system. Individual cables are run between the hub and each controller.

Crossovers are made internal to the hub. Therefore, in a single hub system, all connections are straight-through. Remember that for 10Base-T, each cable connection is limited to 100 meters in length.



4.11 Ethernet Connection Guide

Ethernet 10Base-T is a flexible, low cost method of cabling local area networks. Pointe Controller units must be connected using 10Base-T compatible products. All Ethernet 10Base-T implementation details are defined by the EIA/TIA standard 568A. This standard specifies UTP, an acronym for Unshielded Twisted Pair cable, to be between all nodes on a given 10Base-T network. UTP cables are rated according to their data-carrying ability (bandwidth) and rated by “category” number. The standard specifies category 3, 4, or 5 cable may be used with Ethernet 10Base-T applications. IEEE Ethernet standards limit cable length between nodes to 100 meters (328 feet). The distance limitation is based on the maximum cable signal loss of 11.5 decibels between the source and destination. Due to emerging high speed standards and product capabilities, many sites now install UTP category 5 type cables exclusively. We recommend category 5 cable for all Pointe Controller connections.

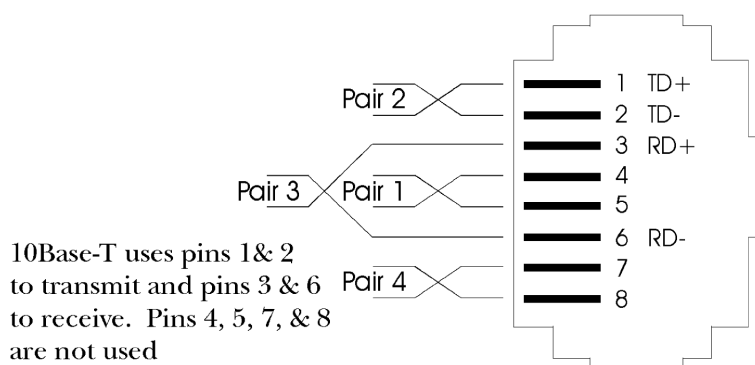
4.11.1 UTP Cable Characteristics

Cabling is the foundation of any network; if it’s incorrect or unstable all other communications characteristics will be unreliable. The most critical aspect of UTP cabling is the maintaining of correct conductor pairing throughout the network. Commonly four-pair (8 wire) 24 AWG thermoplastic insulated solid conductor wire with a 100 ohm impedance and total diameter of less than 6.35mm (0.25 inch) should be used with Ethernet 10Base-T networks. To ensure correct pairing, network vendors offer patch cables (straight-through and crossover) which are assembled with connectors.

4.11.2 Cable Connectors

Pointe Controller units interface the network via the standard 8-pin extension port compatible with RJ45 type connectors. RJ45 type connectors are designed to accommodate rounded PVC outer jacket UTP cable. The strain relief for the cable is provided by the part of the RJ45 connector that acts as a wedge against the outer jacket. The wedge is pressed and locked tightly against the cable jacket when the connector is crimped into place.

A 10Base-T RJ45 connection is shown below:

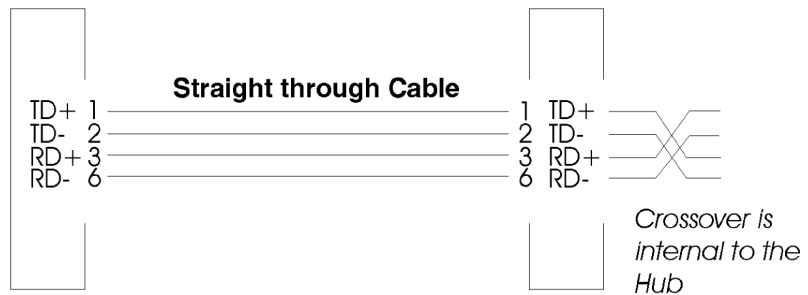


4.11.3 10Base-T Connections

Most hardware ports on Ethernet 10Base-T equipment are wired MDI-X (meaning medium dependent interface crossover) so you can use straight-through cable for interconnecting the network devices. This allows for proper alignment of transmitter and receiver circuits according to 10Base-T networking standards. For hub-to-hub connections, a crossover type cable is commonly required. The figures below illustrate pin assignment and signal names for straight-through and crossover type Ethernet patch cables.

4.11.4 Straight-through Patch Cable

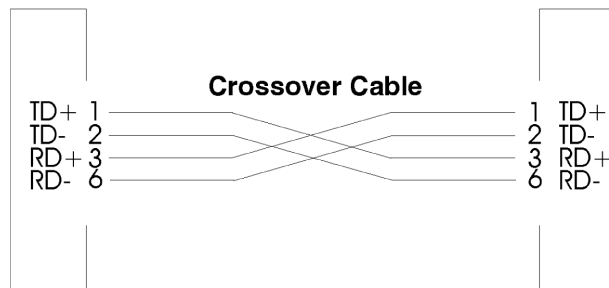
A straight-through cable is commonly used to connect Ethernet 10Base-T devices to a hub. Preassembled patch cables are available from various network product vendors. RJ45 connectors are attached at both ends of an assembled patch cable.



We recommend using a category 5, UTP cable type for all Pointe Controller network connections.

4.11.5 Crossover Patch Cable

Crossover type patch cables are used to connect between hubs or switches. This type of patch cable must also be used for all point-to-point connections, such as a PC-based controller and Pointe Controller unit. Therefore, it is also called a point-to-point cable.



Chapter 5: Developing Controller Programs

Once you have your machine control system designed and your I/O selected and installed, you can proceed with developing the control program itself. Program development is done using the PointeControl Framework application that was installed on your PC from the PointeControl software CD. You can launch the PointeControl Framework from the Windows Start menu by choosing **Start > Programs > PointeControl > Framework**.

Please note that this chapter primarily describes how to develop programs for a single Pointe Controller unit. For more information on developing distributed applications, including enabling Modbus communications and adding remote terminal units (RTUs), please refer to Chapter 8, "Networked Operations," starting on page 216.

TIP: The information provided in this chapter is also available via the PointeControl Framework online help. To access the help, choose **Contents** from the Framework's **Help** menu.

5.1 Basic Concepts in PointeControl

The PointeControl **visual framework editor** (VFE) provides the tools you need to design and compile machine control applications that can be run on Pointe Controller hardware products.

In the framework editor, you create application **projects** made up of individual components — also called “**objects**” — such as Flow Charts, Ladder Diagrams, and Logic Memory tables. Each object’s properties describe instructions and attributes for that object. The framework editor arranges these objects into an expandable hierarchy. You build the project by adding objects to the hierarchy and defining the object properties.

After you complete your application project, you build (compile) the project into runtime module that can be downloaded to and run on your Pointe Controller unit. Using the PointeControl Monitor utility (included with the PointeControl development software), you can monitor and debug your application as it runs on your Pointe Controller.

5.1.1 Multiple Programming Languages

PointeControl provides tools to develop programs in either Nematron’s patented Visual Flowchart Language (VFL) or traditional Relay Ladder Logic (RLL).

You can develop programs using only **Flow Charts** or only **Ladder Diagrams**, or you can mix the two together and use whichever language is most appropriate to each programming task. Charts communicate with each other by reading from and writing to variables in the **Logic Memory** database. Charts never directly reference each other.

5.1.2 Memory Allocation and Access

In traditional control logic engines, program memory is laid out in fixed data tables and tags must be addressed as directly represented variables (DRVs). Sometimes, the user must even manually allocate the available memory.

PointeControl does not have fixed data tables. The user defines tags and variables as needed, using plaintext tag names (or “aliases”) rather than DRVs. Tags are organized by type in the **Logic Memory** database. A broad range of types is supported, including bit, integer, floating point, string, and timer.

The Logic Memory database is globally accessible, allowing all Flow Charts, Ladder Diagrams, and I/O points to communicate freely and continuously with each other using the same data in common. For example, a Flow Chart can read a variable that was set by a Ladder Diagram, or a Flow Chart and a Ladder Diagram can both monitor an input tag that is **associated** with one of the controller’s I/O points, and so on.

All data can be made available to the network via **Modbus mapping**. For more information, see Chapter 8, “Networked Operations,” starting on page 216.

5.1.3 The Scan Cycle

When a PointeControl project is executed on the controller, all of the Flow Charts, Ladder Diagrams, and I/O points that make up the project are “scanned at a specified speed and in a specified order.

Inputs, charts, ladders and outputs are processed in a fixed sequence; each cycle through this sequence is called a logic scan. The general execution sequence is:

1. Get input values from all Pointe Controller modules and save to corresponding tags in the Logic Memory database.
2. Execute all Flow Charts and Ladder Diagrams. Update input and output tags as necessary.
3. Extract output values from Logic Memory database and send to Pointe Controller modules.

Charts and Ladders are run in the order they are placed in the [scan list](#). Only charts and ladders in the list will be executed. You can mix Flow Charts and Ladder Diagrams in the list to get the order you want. For example, if a ladder program is processing inputs and generating values to be processed by a chart, place the ladder first in the list followed by the chart.

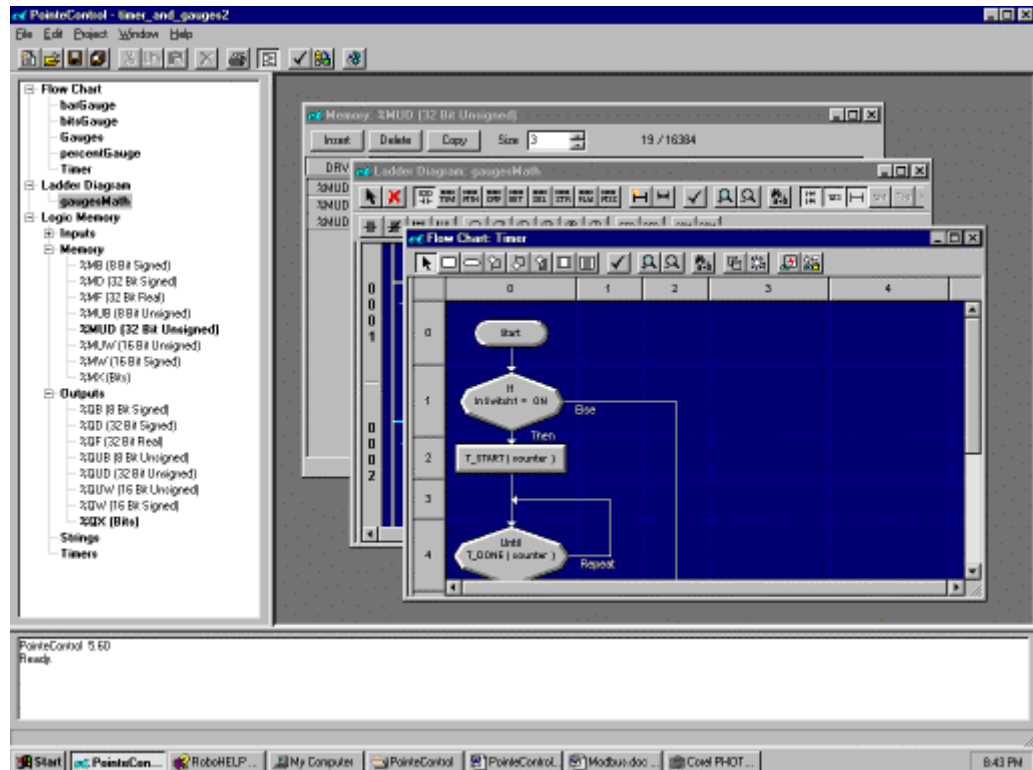
To configure the interval for scanning the Flow Charts and Ladder Diagrams, see [“Setting your project's scan interval”](#) on page 180.

To configure the scan intervals for individual I/O modules and operator panels, see [“Specifying your installed hardware”](#) on page 121.

NOTE: It is possible to create a project that scans so quickly, using up so much processor power on the controller hardware, that the PointeControl Monitor cannot reattach to the controller. For more information, see [“Selecting and attaching a controller”](#) on page 185.

5.2 The Visual Framework Editor (VFE)

With the framework editor you create projects. PointeControl projects provide the instructions to implement a specific machine control sequence. You can make minor or full-scale changes as the control application undergoes periodic change or adjustment.



The framework editor follows Windows NT/2000 user interface conventions to set up projects. If you are new to Windows NT/2000, you should learn some basic Windows skills before starting PointeControl. You can access online help for Windows from the Start menu on your desktop.

Framework Editor Tools

With the framework editor's tools you create the objects, like Flow Charts and Ladder Diagrams, that correspond to interrelated tasks and the logic that refers to variables and I/O devices. The framework editor contains several integrated windows and workspaces. When you click a toolbar, menu selection, or other control, the Editor opens a window for an object or performs a function on an object or the entire project. You define object properties through a variety of drop-down lists, dialog windows, keystrokes, and field entries.

If you are not sure of the action a control performs, you can get a description by placing the mouse cursor over the control. The description appears at the bottom of the Editor window.

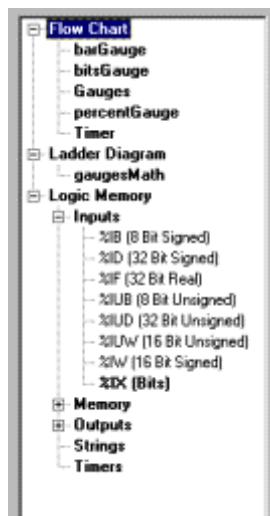
5.2.1 The Framework Editor toolbar



The *framework toolbar* is located at the top of the main Framework Editor window. The buttons on the toolbar perform the following functions, from left to right:

<p>New Object Open Object Save Object Save All Objects</p>	<p>These four buttons manage Flow Chart and Ladder Diagram objects in the project workspace pane. [pages 131, 163]</p>
<p>Cut Copy Paste Delete</p>	<p>These four buttons cut, copy, paste, or delete the currently selected object (from the project workspace) or block (from the currently open Flow Chart or Ladder Diagram). [pages 143, 168]</p>
<p>Print Object</p>	<p>This button prints the object (Flow Chart, Ladder Diagram, or Logic Memory table) that is currently selected in the project workspace pane. [page 112]</p>
<p>Project Workspace</p>	<p>This button shows/hides the project workspace pane located on the left side of the framework editor. [page 109]</p>
<p>Check Integrity Build Runtime</p>	<p>These two buttons check the structure and syntax of your project and then compile a finished runtime program to run on the Pointe Controller unit. [page 181]</p>
<p>Activate Monitor</p>	<p>This button launches the PointeControl Monitor utility, which is used to load, run, and debug compiled programs on the Pointe Controller unit. [page 182]</p>

5.2.2 The Project Workspace pane



The *project workspace pane* is located at the upper left side of the main framework editor window. It keeps track of all of the Flow Charts, Ladder Diagrams, and Logic Memory tables that you create for your PointeControl project.

Add and open resources with the buttons on the toolbar. Resources appear in the project workspace as windows. You can open multiple windows at once, which creates many active windows and dialogs in the editor. Selections from the Window menu cascade or tile these windows. Many resource windows have their own set of tools. For example, predefined blocks, selected from the *Flow Chart toolbar*, allow you to point-and-click to create a chart's

structure. You can then access each block's properties through a floating Block Properties box.



NOTE: You can hide/show this pane by clicking the **Project Workspace** button in the **toolbar**. You can also adjust its size by dragging the right and bottom edges of the pane.

5.2.3 The Object Editor pane

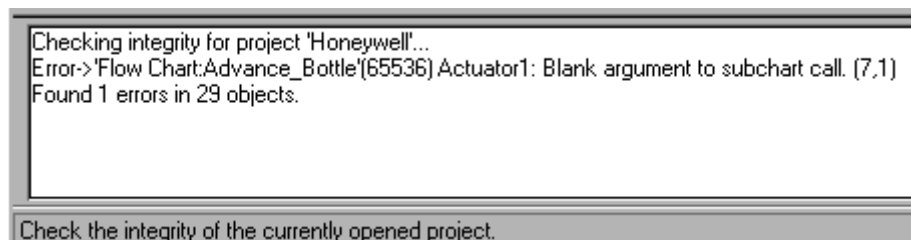
The *object editor pane* is a multi-function area that takes up most of the **framework editor**. When you do not have any objects open, the pane is an empty grey space. When you select an object (i.e., a Flow Chart, a Ladder Diagram, or a Logic Memory data table) from the **project workspace** and open it for editing, that object's editor window appears in the pane.

Each object editor window — including its layout and toolbar — is specific to its corresponding object type. For more information, see also:

- Defining variables in Logic Memory [page 113]
- Navigating the Flow Chart editor [page 132]
- Navigating the Ladder Diagram editor [page 163]

5.2.4 The Messages pane

The *message pane* is located at the bottom of the main **framework editor window**. It displays the progress and error messages that are generated when you check the integrity of your project or build it as a runtime program.



NOTE: You can adjust the size of the pane — or even hide it completely — by dragging the top edge of the pane up and down.

5.3 Managing PointeControl Projects

A PointeControl *project* is a collection of [Flow Charts](#), [Ladder Diagrams](#), [Logic Memory tables](#), [I/O configurations](#), [Modbus mappings](#), and assorted [preferences](#) that, when compiled, implements a specific machine control program on the Pointe Controller unit.

The default working directory for PointeControl project files is:

```
C:\Program Files\Nematron\PointeControl\Projects
```

You can also [import and export projects](#) to other directories.

5.3.1 Creating and opening projects

New Project

To create a new PointeControl project:

1. From the **File** menu, choose **New Project**. The New Project dialog window appears and prompts you to enter a project name.
2. Enter a project name and click **OK**.

Open Project

To open an existing PointeControl project located in your working directory:

1. From the **File** menu, choose **Open Project**. The Open Project window appears.
2. Choose the project you want to open from the drop menu.
3. Click **OK**.

NOTE: If another project is already open, PointeControl will automatically save and close it before creating or opening another.

If the project you want to open is not listed in the drop menu, then it is not located in your working directory. You may need to [import the project from another drive](#) before you can open it.

5.3.2 Importing and exporting projects

Import Project

To import a PointeControl project from another drive to your working directory:

1. From the **File** menu, choose **Import Project**. The [Import Project window](#) appears.

2. In the Source box, select the Drive where the project you want to import is located.

NOTE: The project must be located in a directory named `OCProjects` on that drive. (For example, `A:\OCProjects`.)

3. Choose the project you want to import from the Projects list.
4. Click **Import** to copy the project to your working directory.

Export Project

To export a PointeControl project from your working directory to another drive:

1. From the **File** menu, choose **Export Project**. The **Export Project window** appears.
2. In the Source box, choose the project you want to export from the Projects list.
3. In the Destination box, select the Drive to which you want to export the project.

NOTE: The project will be exported to a directory named `OCProjects` on that drive. (For example, `A:\OCProjects`.) If the directory does not exist, it will be created.

4. Click **Export** to copy the project to the other drive.

5.3.3 Documenting your project

To print the various objects, including database and memory configurations, of the current project, choose **File > Print Project**. You will receive a printed copy of all Flow Charts, Ladder Diagrams, and Logic Memory tables from the selected printer.

NOTE: To print a single Flow Chart, Ladder Diagram, or Logic Memory table, open the object from the **Project Workspace** pane and choose **File > Print**.

To create and print a cross-reference report of the location of every instance of all Logic Memory tags and variables used in the current project, choose **File > Print Cross Reference**. You will receive a printed copy of a formatted cross-reference report from the selected printer. The format includes:

- The name (Alias) of the tag or variable;
- The chart name and block coordinates for each instance of the tag; and
- A description of each usage of the tag.

5.4 Defining Variables in Logic Memory

In PointeControl, *Logic Memory* is the database of all tags, variables, and data structures used in your program. Logic Memory is globally accessible, allowing all Flow Charts, Ladder Diagrams, and I/O points to communicate freely and continuously with each other using the same data in common. For example, a Flow Chart can read a variable that was set by a Ladder Diagram, or a Flow Chart and a Ladder Diagram can both monitor an input tag that is associated with one of the controller's *I/O points*, and so on.

Logic Memory supports the following data types:

- An **Input** tag is a bit, integer, or real number variable that is associated with a Pointe Controller input channel.
- A **Memory** tag is a bit, integer, or real number variable that is stored in temporary memory.
- An **Output** tag is a bit, integer, or real number variable that is associated with a Pointe Controller output channel.
- A **String** is used to store ASCII text. Strings can also be associated with *operator panel* display lines.
- A **Timer** is a special data structure that is used to count real time in milliseconds, based on the Pointe Controller's internal clock rather than on the project's *scan cycle*.

As you define tags and variables in Logic Memory, keep in mind which tags you will need to associate with to the Pointe Controller's I/O points and which you will need to facilitate your program's internal logic flow.

NOTE: All tags and variables are defined using plaintext names or "aliases." DRVs and wire labels are not used in PointeControl.

5.4.1 Java reserved words

Since your PointeControl project will ultimately be compiled into Java classes, you cannot use any of Java's "reserved words" as aliases in Logic Memory. Reserved words are terms that have their own inherent functions within the programming language itself and therefore can conflict with similarly named variables.

PointeControl will automatically check for conflicts whenever you attempt to *compile your project*, but it is better to avoid using reserved words when defining your variables in the first place.

Java's reserved words include:

abstract	default	goto	operator	synchronized
boolean	do	if	outer	this

break	double	implements	package	throw
byte	else	import	private	throws
byvalue	extends	inner	protected	transient
case	false	instanceof	public	true
cast	final	int	rest	try
catch	finally	interface	return	var
char	float	long	short	void
class	for	native	static	volatile
const	future	new	super	while
continue	generic	null	switch	

Other reserved words in PointeControl: Chart, Project.

5.4.2 Defining Input, Memory, and Output tags

Input, Memory, and Output tags are all basically the same type of data structure. Individual tags differ from each other in only two ways:

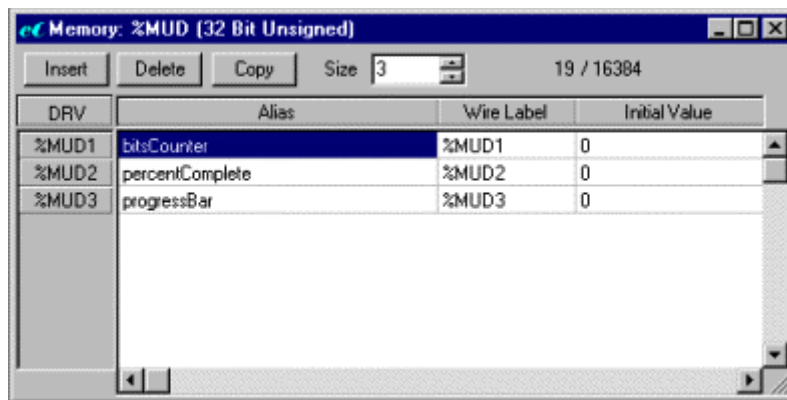
- **How the tag is used** – Input and Output tags are typically associated with the various **I/O points** on the Pointe Controller unit. Memory tags are used as “scratch values within the program itself and typically do not have any direct readout.
- **What size the tag is** – Every tag must be defined as a specific register size and numeric mode. The size/mode of the tag determines the range of values the tag can have:

SIZE/MODE	VALUE RANGE
Bit/Boolean	0 or 1
8-bit Unsigned	0 to 255
16-bit Unsigned	0 to 65,535
32-bit Unsigned	0 to 4,294,967,295
8-bit Signed	-128 to 127
16-bit Signed	-32,768 to 32,767
32-bit Signed	-2,147,483,648 to 2,147,483,647
32-bit Real	-3.4 x 10 ³⁸ to 3.4 x 10 ³⁸

Before you define any new tags, you should have some idea of what types and sizes are required by your control application. You can always define more tags later, but it makes for better application design to plan your tags in advance.

To define new Input, Memory, or Output tags:

1. In the **Project Workspace pane**, double-click on **Logic Memory** to expand the hierarchy.
2. Double-click again on the desired data type: **Inputs**, **Memory**, or **Outputs**. The data type will be expanded to show all of the individual data tables within the type.
3. Select and open the data table that corresponds to the desired tag size/mode. The editor window for that data table will appear. In this example, the 32-bit Unsigned Memory data table is selected.



4. Adjust the **Size** control to add addresses to the data table. Either use the arrow buttons or directly enter a number.

NOTE: As you add more addresses, the data table increases in size and uses more memory. The memory used/available readout shows the number of addresses used by all data tables and the total number of addresses available.

5. Click in the **Alias** field of the first empty address and enter a name for the tag. The name must be a continuous alphanumeric string that does not begin with a number; for example, STA1_UP_LIMIT.

NOTE: Aliases are case sensitive.

6. Click in the **Initial Value** field and enter a value to initialize on program startup. Default initial value is 0.
7. Repeat steps 4 through 6 as needed.
8. When finished, close the editor window. You will be prompted to save your changes.

You can insert and delete addresses in the middle of the table using the Insert and Delete buttons. And since PointeControl refers to all tags and variables only

by Alias, it is not necessary to keep your tags in any particular address or DRV order.

Diagnostics Metadata

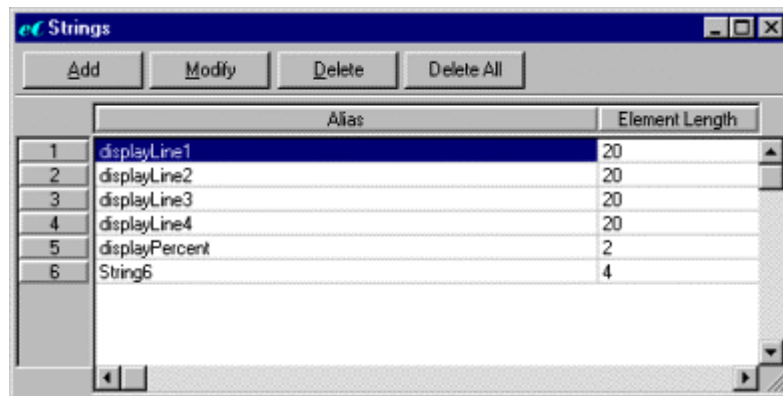
Every Input, Memory, and Output tag has associated with it three metadata registers. These registers are allocated automatically when the tag is first defined in Logic Memory, and they can be used to store diagnostic information about the tag. For more information, see [Diagnostics Commands](#) on page 316.

5.4.3 Defining strings in Logic Memory

String variables are defined using dialogs that allow specification of a name, the size or number of characters the variable may hold, and an initial value.

To define new String variables:

1. In the [Project Workspace pane](#), double-click on **Logic Memory** to expand the hierarchy.
2. Select and open the **Strings** data table. The Strings editor window will appear.

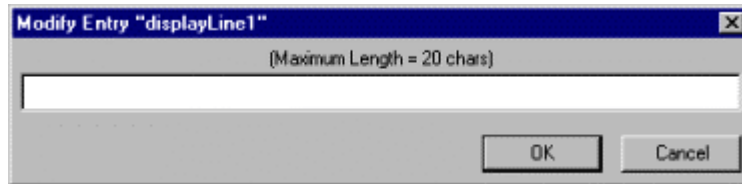


3. Click **Add** to add a new address to the table.
4. Click in the **Alias** field of the empty address and enter a name for the String. The name must be a continuous alphanumeric string that does not begin with a number.

NOTE: Aliases are case sensitive.

5. Click in the **Element Length** field and enter the maximum number of ASCII characters that the String should hold. Maximum length is 255.

6. If you want to set an initial value (i.e., the value to which the String will be set on program startup), click **Modify** and enter the value.



7. Repeat steps 3 through 6 as needed.
8. When finished, close the editor window. You will be prompted to save your changes.

5.4.4 Defining timers in Logic Memory

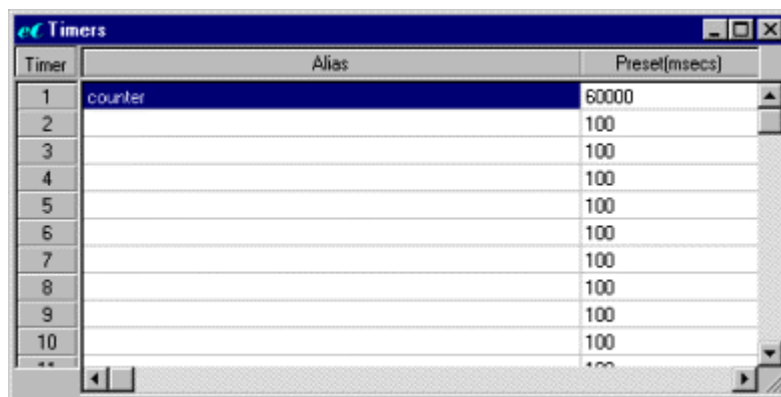
A Timer is a special data structure that is used to count real time in milliseconds, based on the Pointe Controller's internal clock rather than on the project's scan cycle. Up to 2048 Timers are pre-allocated in Logic Memory, and you can use any one of them once you have given it a name.

Each Timer is assigned a Preset value (in milliseconds), either when it is first defined or later by using a function block. The Timer can then be started and stopped as needed by the logic flow, and when it reaches its Preset it sets a "done bit that can be read.

For more information on using Timers, see [Timer Commands](#) (for Flow Charts) on page 303 or [Timer and Counter blocks](#) (for Ladder Diagrams) on page 350.

To define new Timers:

1. In the [Project Workspace pane](#), double-click on **Logic Memory** to expand the hierarchy.
2. Select and open the **Timers** data table. The Timers editor window will appear.



3. Click in the **Alias** field of the empty address and enter a name for the Timer. The name must be a continuous alphanumeric string that does not begin with a number.

NOTE: Aliases are case sensitive.

4. Click in the **Preset** field and enter timer preset (in milliseconds).
5. Repeat steps 3 and 4 as needed.
6. When finished, close the editor window. You will be prompted to save your changes.

5.4.5 Importing and exporting databases

As an alternative to defining every **Logic Memory** tag individually, you can import a pre-made tag database into PointeControl from an external file. You can also export a existing project's Logic Memory tables to an external database file for backup or future use.

Database File Format

To import a database into a project, you must first format the database as a delimited text file. PointeControl recognizes three types of text files for import/export:

- *.txt – Fields delimited by tabs.
- *.prn - Fields delimited by spaces.
- *.csv - Fields delimited by commas.

WARNING: Attempts to import other, non-supported file types – particularly those that contain non-ASCII characters – can compromise the PointeControl system.

Each line in the text file should describe a single Input/Memory/Output tag, String, or Timer. The format of the line varies according to the type of variable being described, as explained in the table below. Note that in the Format column, the dash (—) represents a delimiter. Be sure to use the delimiter appropriate to the file type.

TYPE	FORMAT	DESCRIPTION
Input, Memory, Output	<i>DRV—Alias—Wire—Value—Retain</i>	<p><i>DRV</i> – Address describing the type of tag and its position in the Logic Memory tables. (For example, “QX3” is an Output Bit, table position 3.) Must be consecutively numbered. Mandatory.</p> <p><i>Alias</i> – The common name by which the tag is referred in PointeControl. Up to 30 characters. Mandatory.</p>

		<p><i>Wire</i> – The tag’s wire label. Maximum of 10 characters. Must be enclosed in tildes (~xxx~). Optional, not used in PointeControl.</p> <p><i>Value</i> – The initial value of the tag upon program start. Optional.</p> <p><i>Retain</i> – A yes/no option to specify whether the tag should be retained in retentive memory. Optional, not used in PointeControl.</p>
String	STRING — <i>Alias</i> — <i>Length</i> — <i>Value</i>	<p>STRING – Denotes that the line describes a String. Mandatory.</p> <p><i>Alias</i> – The common name by which the String is referred in PointeControl. Can be up to 30 characters. Mandatory.</p> <p><i>Length</i> – The element length of the String. Maximum of 255. Mandatory.</p> <p><i>Value</i> – The initial value of the String upon program start. If defined, must be less than or equal to Length and must be enclosed in quotes (“xxx”). Optional.</p>
Timer	TIMER — <i>ID</i> — <i>Alias</i> — <i>Preset</i>	<p>TIMER – Denotes that the line describes a Timer.</p> <p><i>ID</i> – The Timer ID (table position) to be defined.</p> <p><i>Alias</i> – The common name by which the String is referred in PointeControl. Can be up to 30 characters. Mandatory.</p> <p><i>Preset</i> – The preset value of the Timer, in milliseconds. Optional.</p>

Notes:

- If an optional field is not defined in a .csv file, the comma delimiter is used as a place keeper for that field. Default values are assigned when no value is specified for a field.
- Since you may enter any number of spaces and tabs as delimiters in .prn and .txt files, you need to observe these rules when building a .prn or .txt database file:
 - If a field following an undefined Wire is defined, the Wire must be indicated by closed tildes (~~).
 - If a field following an undefined value is defined, the undefined value must be indicated by the default value.

TIP: If you still do not understand how the database file must be formatted, try exporting the database from an existing project. You can then compare the exported file against the Logic Memory tables and see how specific tags are described. For more information on exporting database files, see below.

Importing a Database

To import a database file into a PointeControl project:

1. Format the database file as described above.
2. If a project is open in PointeControl, close it (**File > Close Project**). If you do not close the project, you will not be able to import to it.
3. From the **File** menu, choose **Import Database**. The Import Database dialog will appear.
4. Under **File name**, enter the file name of the database file to be imported. If necessary, click **Browse** to find the file on your drive.
5. Under **Project name**, select the project from the drop-down menu.
6. Click **Import**.
7. Specify whether you want the imported database to overwrite the project's existing Logic Memory or merge with it. (Click **Yes** to overwrite, **No** to merge.)

When importing a text database file, if PointeControl detects an erroneous line or duplicate Aliases, you have the options of discarding the line or canceling the import process. If PointeControl detects a duplicate Alias in a definition, PointeControl converts the name to blanks.

Exporting a Database

To export a database file into a PointeControl project:

1. If a project is open in PointeControl, close it (**File > Close Project**). If you do not close the project, you will not be able to export from it.
2. From the **File** menu, choose **Export Database**. The Export Database dialog will appear.
3. Under **Project name**, select the project from the drop-down menu.
4. Under **File name**, enter the file name of the database file to be exported. If necessary, click **Browse** to find the save directory on your drive.

NOTE: PointeControl will automatically format the file according to the file suffix you use: *.txt, *.prn, or *.csv.

5. Click **Export**.

5.5 Associating Tags with I/O points

I/O points are the many input and output channels that are made available to your control application when you install OptiLogic I/O modules and operator panels in your Pointe Controller unit. Each point is associated with a tag in your project's **Logic Memory** database. Your project controls these points by reading from and writing to the associated tags.

To configure your project's I/O points, you must first specify which modules and panel are actually installed in your controller. Then you can step through each module and manually associate Logic Memory tags to each specific input and output channel.

NOTE: Each Logic Memory tag can be associated with only one I/O point, so make sure that you have defined enough tags to cover every point on your installed modules and operator panel. If necessary, you can go back and define additional tags as you configure each module.

5.5.1 Specifying your installed hardware

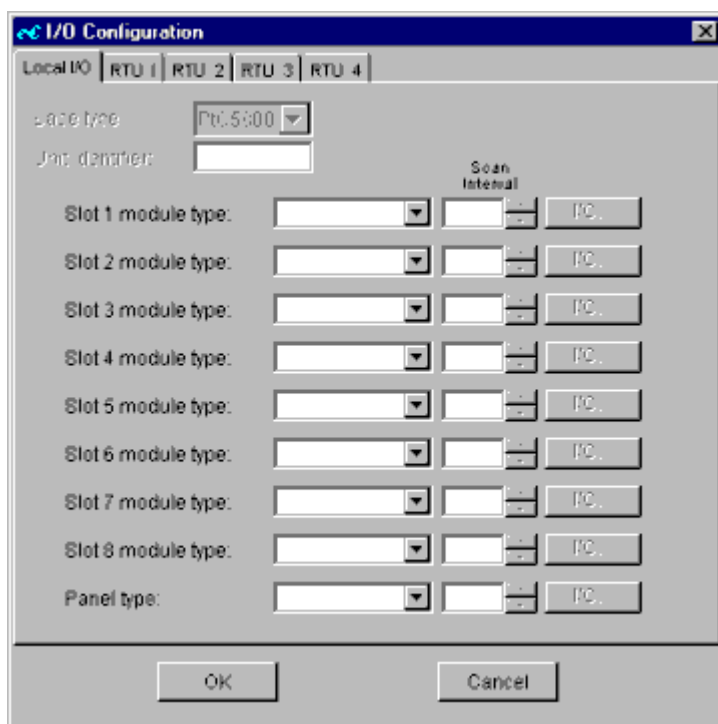
Before you can associate Logic Memory variables with individual I/O points, you must first specify which OptiLogic I/O modules and operator panel will ultimately be installed in your Pointe Controller unit. Your PointeControl project cannot recognize or communicate with these modules at runtime if they are not properly configured.

NOTE: You can add and remove modules at any time so long as you properly configure them as described here, then adjust your programming to accommodate the changes and **recompile** your project for the Pointe Controller unit.

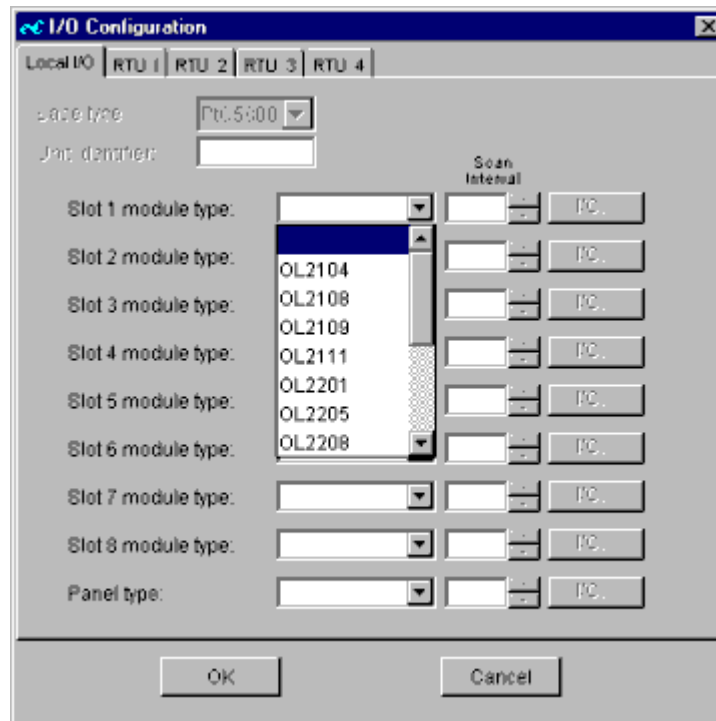
To specify what I/O modules and operator panels are installed in your Pointe Controller unit:

1. Check your controller and make note of what modules you have installed. (For more information on selecting and installing modules, see Chapter 4, "System Design and Installation," starting on page 70.)

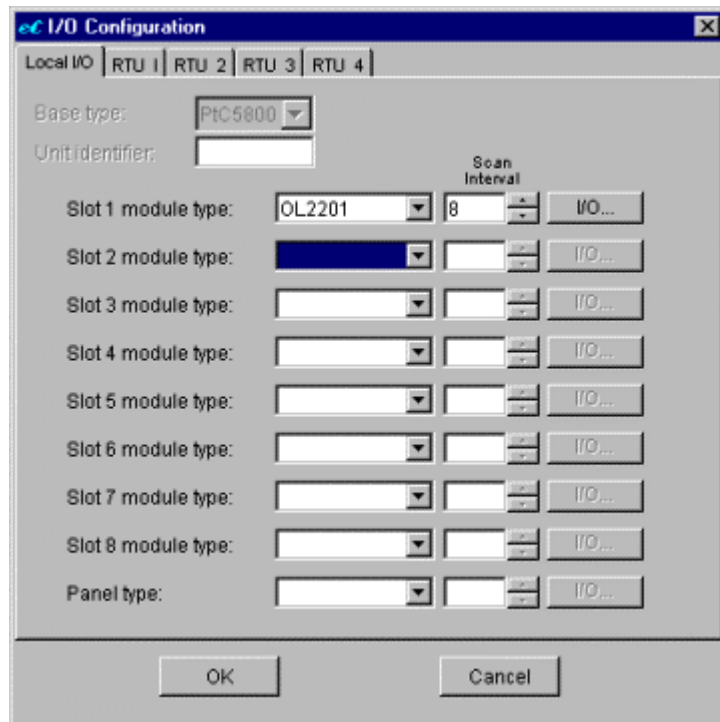
- From the **Project** menu, choose **Configure I/O**. The I/O Configuration window appears.



- Starting with **Slot 1 module type**, click on the drop-down menu to get a list of available I/O modules.



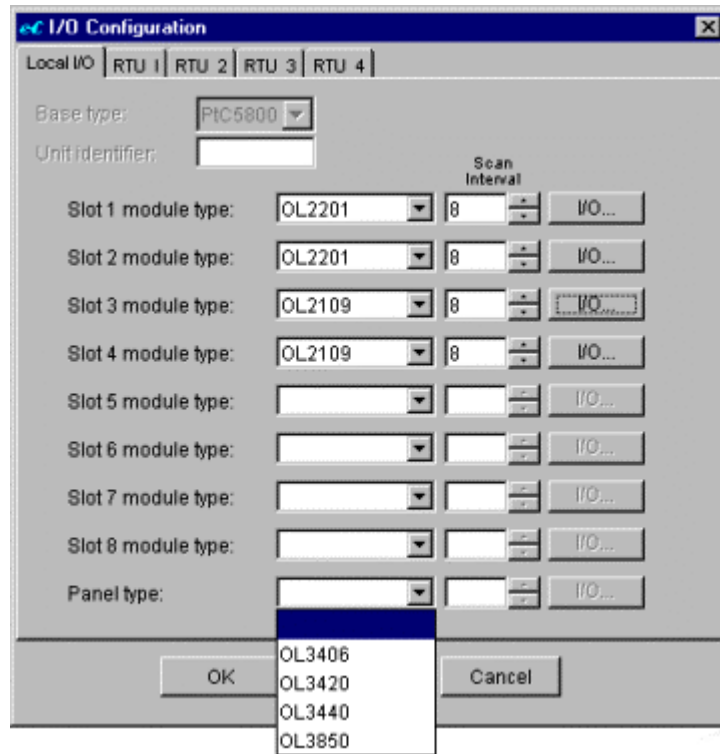
4. Select the model number that corresponds to the I/O module that is actually installed in the first slot in the Pointe Controller unit. In this example, an OL2201 Digital Input module is selected.



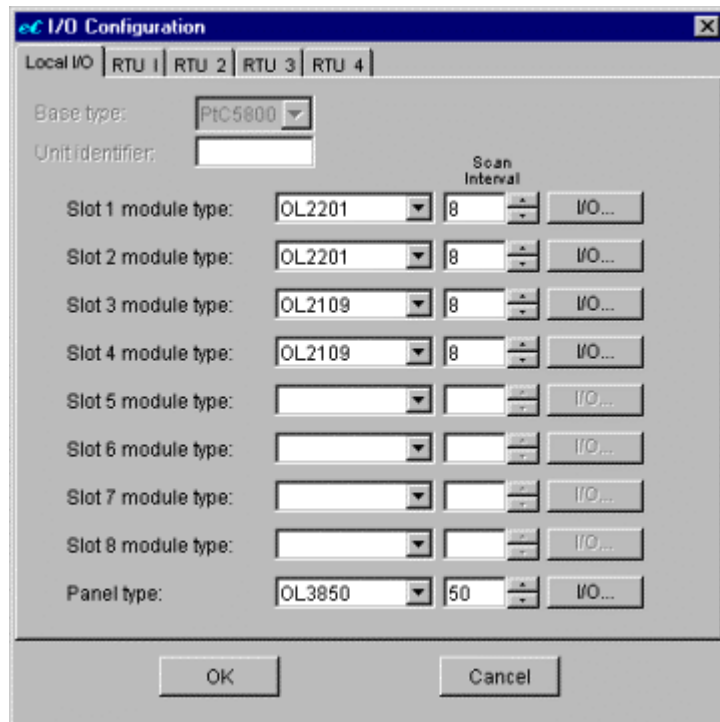
5. Continue through the rest of the slots (1 through 8), selecting the model numbers that correspond to the installed modules. If no module is installed in a given slot, then skip it.

NOTE: To clear any selection, select the <blank> option at the top of the drop-down menu.

6. To the right of Panel type, click on the drop-down to get a list of available operator panels. This list will be different from the list of I/O modules above.



7. Select the model number that corresponds to the operator panel, if any, that is actually connected to your Pointe Controller unit. In this example, an OL3850 Operator Terminal is selected.



8. Proceed to [Configuring I/O modules](#), or click **OK** to save your changes and close the window.

Scan Intervals and Scanner Overload

Each installed I/O module has a **Scan Interval** which determines how frequently the module is **scanned** by the Pointe Controller unit. In most cases, the default values should be used. However, if you encounter performance issues while running a compiled program, you may need to adjust the values in order to tune processor and memory usage.

Also, if you install modules that require extremely low Scan Intervals (for example, the OL2602 has a default Scan Interval of 3), then you may be warned of a Scanner Overload. To avoid the overload, you must either increase the modules' Scan Intervals or redesign your project to use fewer or different modules.

For more information, see Chapter 4, "System Design and Installation," starting on page 70.

5.5.2 Configuring I/O modules

For each I/O module that you have **specified** as being installed in your Pointe Controller unit, you must configure that module’s individual settings and I/O points to work with your project. The configuration options are accessed by clicking on the **I/O** button to the right of the module.

NOTE: All of the tags for a given module must already be defined in Logic Memory *before* you can associate them with the module’s I/O points. If the tags are not defined or if you’re not sure what tags are required, review the module below and then go back to **Defining Variables in Logic Memory**.

Select an OptiLogic I/O module to configure:

MODULE	GO TO...
OL2104 Relay Output Module	Page 233
OL2108 Relay Output Module	Page 236
OL2109 DC Sinking Output Module	Page 240
OL2111 AC Solid-state Relay Module	Page 244
OL2201 Digital Input Simulator Module	Page 248
OL2205 AC/DC Digital Input Module	Page 250
OL2208 DC Digital Input Module	Page 254
OL2211 AC Digital Input Module	Page 256
OL2252 Dual Pulse Counter Module	Page 261
OL2258 High Speed Counter Module	Page 268
OL2304 Analog Voltage Output Module	Page 273
OL2408 Analog Voltage Input Module	Page 276
OL2418 Analog Current Input Module	Page 279
OL2602 Dual Serial Port Module	Page 282

For complete technical descriptions of all of these modules, see Appendix A, “OptiLogic Technical Specifications.”

When you have configured all of the I/O modules installed in your Pointe Controller unit, proceed to **Configuring operator panels**.

5.5.3 Configuring operator panels

If you have **specified** an operator panel as being connected to your Pointe Controller unit, you must configure its individual settings and I/O points to work with your project. The configuration options are accessed by clicking on the **I/O** button to the right of the panel.

NOTE: All of the tags for a given panel must already be defined in Logic Memory *before* you can associate them with the panel's I/O points. If the tags are not defined or if you're not sure what tags are required, review the panel below and then go back to [Defining Variables in Logic Memory](#).

Select an operator panel to configure:

MODULE	GO TO...
OL3406 Pushbutton/Indicator Panel	Page 284
OL3420 Operator Terminal	Page 288
OL3440 Display Panel	Page 291
OL3850 Keypad Terminal	Page 292

For complete technical descriptions of all of these panels, see Appendix A, "OptiLogic Technical Specifications."

5.5.4 Configuring additional OptiLogic RTUs

You can configure up to four additional OptiLogic Remote Terminal Units (RTUs) to work with your Pointe Controller unit. These units are slaved to your controller using the OptiLogic UDP/IP communication protocol.

For more information, see Chapter 8, "Networked Operations," starting on page 216.

5.6 Building and Editing Flow Charts

Flow Charts depict various types of information and control processing problems and their means of solution. Charts consist of symbols having a given signification, brief explanatory text, and connecting lines. Each symbol relates to an unambiguous and meaningful name that is consistent throughout the charts. The connecting lines show the path of execution through the chart.

Multiple Flow Charts

A single Flow Chart ideally performs a single task; however, a total system solution requires the execution of multiple tasks, often simultaneously. PointeControl solves this problem by providing concurrent execution of any number of separate Flow Charts. With multiple charts executing concurrently, each chart focuses only on its specific task, which greatly simplifies the chart structure.

The PointeControl Framework creates a Chart List that specifies which charts are executed and the order in which the charts are executed. At startup, the Pointe Controller unit performs all initialization procedures then begins the normal scan cycle. During each cycle, the system performs an input scan to read data from the configured modules, executes each chart specified in the Chart List, and performs an output scan to write data to the modules.

All charts execute during each cycle — starting with the first chart in the chart list and proceeding through the entire list. In the most basic cycle, the first chart in the list runs, then the second chart in the list, then the third, until all charts have run. However, in many real world situations, the chart being executed reaches a block where it must wait for some event to occur before it can proceed. While waiting, the chart (typically) yields control to the next chart in the list, allowing that chart and all other charts to continue executing as normal. During the next execution cycle, if the event has occurred, the chart continues execution from that point; otherwise, the chart again yields control to the next chart.

Program and Subcharts

The PointeControl Framework provides two levels of flowchart development: Program and Subchart. Program charts describe the steps needed to perform a particular control task or process. The more precise a Program chart, the easier it is to develop function flowcharts.

Subcharts contain the more detailed information necessary to make the chart function. You can use these powerful reusable subprograms many times in several different flowcharts.

Flow Chart Blocks

Flow charts contain distinct block types: Process, Terminator, Decision blocks, and Subcharts.

Rectangles with the entry point at the top and the exit point at the bottom represent **Process** blocks:

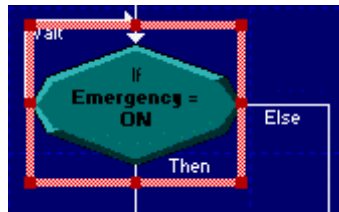


Each Process block contains a description of the action or actions to be taken. The same Process block can contain multiple commands, with each command executing sequentially.

Terminator blocks define the beginning and ending of a chart's program flow: All Flow Charts have a Start and an Exit or Return terminator block. Subcharts return to the calling chart at a Return block.

TIP: A Flow Chart can have only one Start block, but may have more than one Exit or Return block.

Diamonds, with an entry point located at the top and exit points on the right side and at the bottom, represent **Decision** blocks:



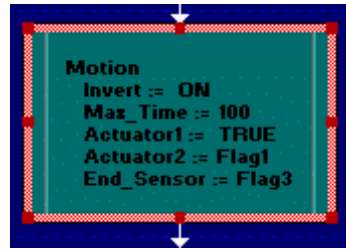
Decision blocks include Condition, Repeat/Until Loop, and While Loop blocks. Any yes/no question can be asked within these blocks. If you need to test two conditions at the same time, you can describe both within the test block by using the logical AND, OR, XOR, and NOT operators. For example:

```
START_PB = ON AND EMERGENCY = OFF
```

With the Repeat/Until Loop and the While Loop blocks, you can design a block that repeats commands. The Repeat/Until Loop continues to ask the questions contained in its block until its conditions become true. The While Loop continues to ask the questions contained in the block until its conditions are no longer true.

Flow Charts are event-driven diagrams that do not depend on time constraints, although you can assign a time-out to a decision block. When the timeout-defined number of microseconds passes, the program proceeds to the Else path, even if the block has not fulfilled all of its conditions.

With **Subchart** blocks you can add calls to other charts:



You can place multiple Subchart blocks in a single Flow Chart.

5.6.1 Creating a new Flow Chart

To create a new Flow Chart:

1. In the Project Workspace pane, select **Flow Chart**.
2. Click the **New Object** tool in the Framework Editor toolbar, or choose **New** from the **File** menu. A new Flow Chart with a default name (ChartN) will be added to the project hierarchy.

To open a Flow Chart for editing, simply double-click on it or select the chart and click the **Open Object** tool in the Framework Editor toolbar. When you open an chart, its editor window will appear in the Object Editor pane.

Saving a Flow Chart

All program objects are saved automatically whenever you close them. However, you can save a Flow Chart while it's open by clicking the **Save Object** or **Save All Objects** tool in the Framework Editor toolbar.

Deleting a Flow Chart

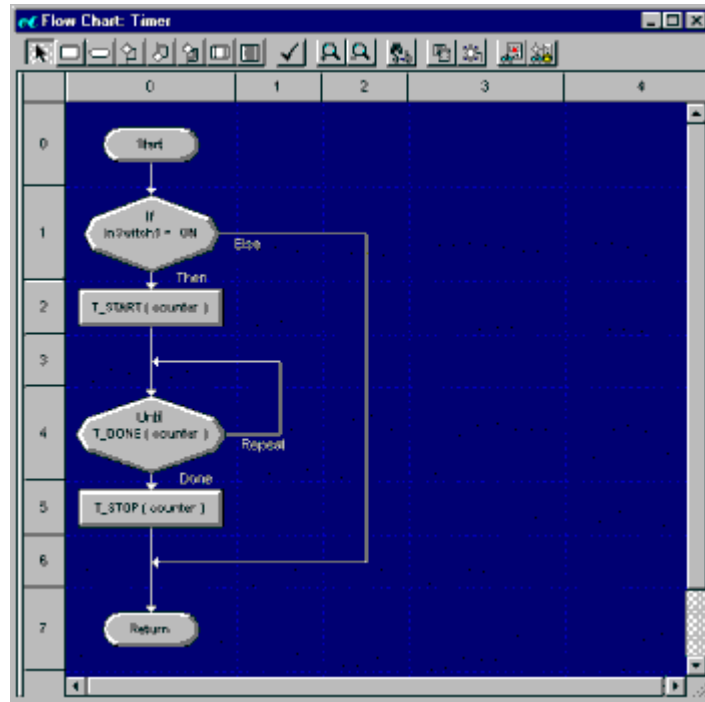
You can delete Flow Charts in the same place you open them: the Project Workspace pane. To delete a Flow Chart:

1. Close all open Flow Charts, Ladder Diagrams, and Logic Memory tables.
2. Select the desired Flow Chart in the Project Workspace pane. (You may need to expand the hierarchy to select it – double-clicking expands it, double-clicking again collapses it.)
3. Click the **Delete Object** tool in the Framework Editor toolbar, or choose **Delete** from the **Edit** menu.

NOTE: Before deleting a Flow Chart, PointeControl will ask for a confirmation.

5.6.2 Navigating the Flow Chart editor

The Flow Chart editor window includes a work area and the tools you need to create an executable chart. With the window's special set of tools, you add and define the blocks that represent the program flow of interrelated tasks.



Object-oriented flowcharting gives you freedom to easily establish your chart's structure. With the arrow cursor and tool bar, you can place and manipulate Flow Chart blocks. By placing blocks you direct program flow, link to subcharts, and add comments to the chart. You can select blocks and perform cut/copy, paste, delete, and resizing functions to a selected block.

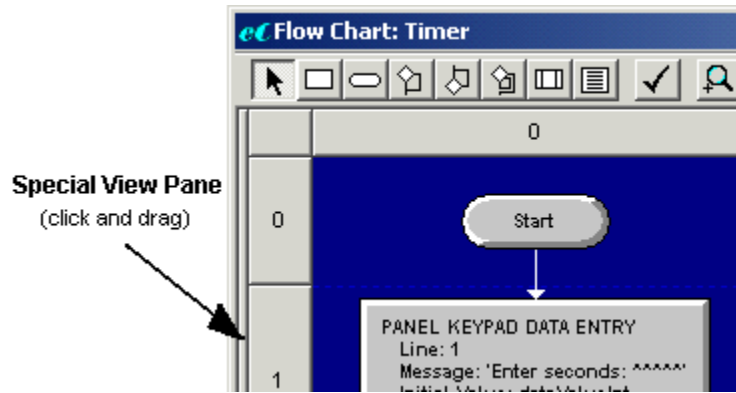
Flow Chart Workspace

This area provides an object-oriented graphic screen where you place and edit blocks. The Workspace uses a grid with vertical and horizontal coordinate labels at the left and top of the workspace. Each grid coordinate does not reflect any absolute measurement, but indicates block locations (which can vary in size). A new Flow Chart has a start block at 0 (h), 0 (v) and a return or exit block at 0,1. The start block's properties use default values. For example, a new chart's name appears as Chart [#], until you change it.

Special View Pane

To reveal a special view pane:

1. Pass the cursor over the gray bar (the adjust pane size) between the left side of the window and the vertical coordinates. The cursor changes to a black double-headed arrow that points left and right.

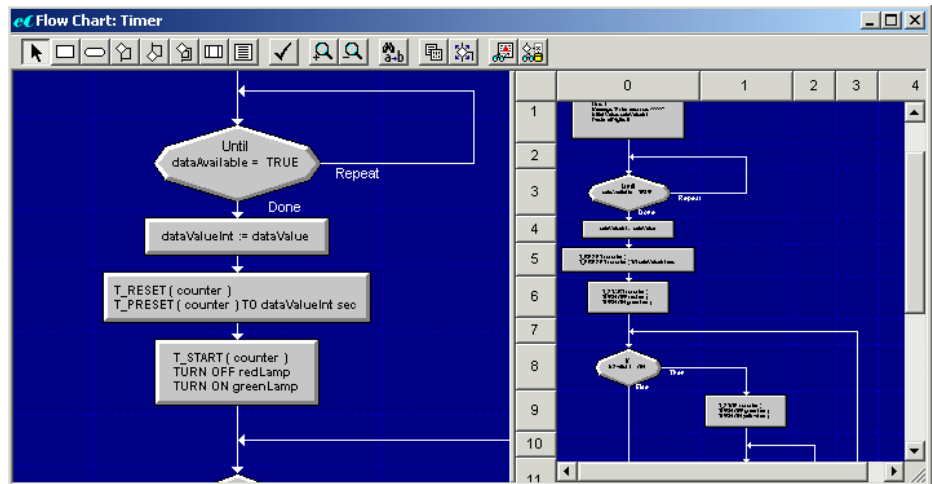


2. Click and drag the adjust pane size bar to the right. The special view pane appears as the workspace slides to the right.

TIP: If the entire Flow Chart editor window moves, you have grabbed the left side of the window, not the adjust pane size bar. Move the cursor slightly to the right and try again.

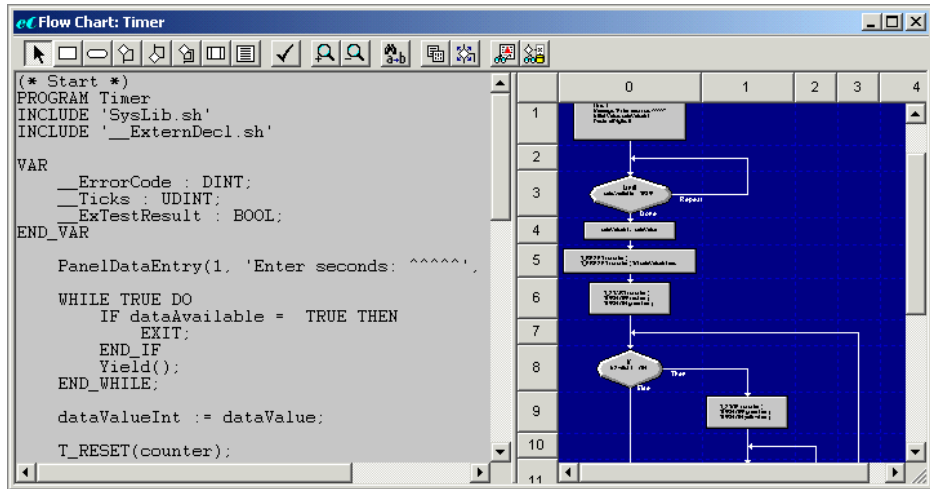
To determine the view pane display, right-click in the special view pane and select Chart View, ST Code View, or Hide from the drop-down menu:

- When you select **Chart View**, the chart appears in the special view pane at full magnification level. The view scrolls automatically to keep the display in the view pane centered around the current position of the mouse cursor in the workspace window:



This option allows you to display smaller magnification levels in the workspace window and the area around the cursor at full magnification in the special view pane. You may find this feature useful with a large chart, since you can see the entire chart and a smaller area of detail at the same time.

- When you select **ST Code View**, a representation of the chart appears in IEC-1131 structured text source code:



The editor automatically updates the code view as you make changes to the chart.

- When you select **Hide**, the special view pane no longer appears.

The Flow Chart toolbar



Access the Flow Chart programming tools from this bar. The toolbar allows you to select blocks, place a variety of block types, change zoom levels and special viewing mode, hide/show block labels, and check chart integrity. To select a tool, you click its button.

Select Tool	With the arrow cursor you can select, move, and resize blocks. [page 143]
Process Block Terminator Block Condition Block Repeat/Until Loop Block While Loop Block Subchart Block	The six Block tools determine block type placed in a Flow Chart. [page 135]
Comment Tool	With the Comment Tool, you can add notes to your Flow Chart. [page 145]
Check Chart Integrity	This button checks the structure and syntax of the Flow Chart. [page 181]

Zoom In Zoom Out	The Zoom In and Zoom Out tools expand and reduce the image of your Flow Chart in the workspace, making it easier to review large sections of the logic flow at a glance. [page 177]
Replace Text	The Replace Text tool finds and replaces all instances of specified text within the current chart. [page 177]
Toggle Labels	The Toggle Labels tool turns on/off the display of block Captions in the editor window. (Captions can be added to blocks via the Block Properties window.)
Size to Content	The Size to Content tool resizes all of the blocks in the chart to fit their respective contents. This makes it easier to read the chart and see what blocks do without opening them one by one.
View Change History	This tool displays the history of changes to the current Flow Chart. [page 146]
View Tag Cross Ref	This tool displays a cross-reference of what tags and variables are used in the chart. [page 178]

5.6.3 Placing and configuring Flow Chart blocks

Object-oriented flowcharting gives you freedom to easily establish your chart’s structure. With the arrow cursor and toolbar, you can place and manipulate Flow Chart blocks. By placing blocks, you direct program flow, link to subcharts, and add comments to the chart.

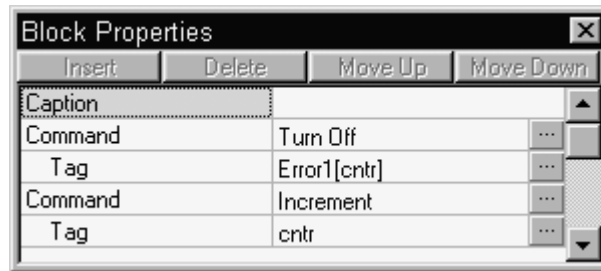
To place a new Flow Chart block:

1. Select a block type from the Flow Chart **toolbar**:
 - Process Block [page 151]
 - Terminator Block [page 154]
 - Condition Block (If/Then/Else) [page 156]
 - Repeat/Until Loop Block [page 158]
 - While Loop Block [page 159]
 - Subchart Block [page 161]
2. Place the selected block type by clicking on an existing flowline in the chart. The block appears at the specified location. Flowlines are automatically redrawn to incorporate the new block into the logic flow.



Block Properties

When you insert a new block into the chart, the editor assigns default properties, based on the block type. You change properties for a block through the Block Properties window, accessed by double-clicking the block. A list of attributes for

the selected block appears in the window; the types of attributes listed depend on the type of block selected.



The window provides editing controls and fields that allow you to specify what actions occur or what conditions are tested when the runtime system executes a block:

- The list buttons modify the command list in **Process blocks** and the argument lists in subchart **Start blocks**. These buttons allow you to **Insert** an item (after the highlighted item), **Delete** the highlighted item, or move the highlighted item **Up** or **Down** in the list. The Command field must be highlighted in order to access the list buttons. Doing this allows you to enter multiple commands in one process block using the Insert button.
- The **Build Expression** () button opens a dialog to **build an expression** for the selected property or action.
- The **Close** button () closes the box. To reopen the box, double-click any block.
- The scroll bars allow you to move through the list of properties.

The Block Properties window always opens at a default size. You may change the size of the currently open box by grabbing a line or corner with the cursor.

5.6.4 Building logical expressions

Most Flow Chart blocks reference a tag or expression. Tags provide the chart access to real device inputs and outputs. With expressions you can assign new values based on numeric calculations that often include references to other tags. Decision blocks refer to tags and expressions that control program flow through a chart.

For all blocks and commands, you enter tags and expressions through a similar dialog box. Although these dialog boxes vary slightly depending on the required response, they all contain a keypad and an area to enter an expression.



The keypad allows you to build an expression using the mouse rather than the keyboard. When you select a keypad button, the Flow Chart editor inserts the button’s corresponding characters at the text insertion point in the selected field. You can add numbers, arithmetic and Boolean operators, logic memory prefixes and data types, spaces, and keyword constants (ON, OFF, TRUE, FALSE). The Backspace button erases the character to the left of the cursor; the Clear button erases the entire field.

A second box, the Select Expression Argument list box, appears beside the tags and expressions dialog box and contains a list of tagnames:

- Add a reference to any defined Logic Memory database element using the tagname list. From the Arg Type list, you can select Inputs, Outputs, Memory database elements, Strings, Timers, Functions, or Local Variables. The tags available for the selected argument type appear in the Selection List. Double-click an item in the selection list to insert the corresponding name in the tag area of the tags and expressions dialog. All tags are displayed by Alias.
- You can manually enter an entire tag reference or expression by clicking in the field on the tags and expressions dialog and typing the required text using the keyboard.
- From the Arg Type list, you can also select **Functions** as an expression argument. When Functions is selected, the View list presents the options Math, String, Timer, and Date/Time.

Math functions:

- **SHL**(value,bits) returns the value shifted to the left by the specified number of bits.
- **SHR**(value,bits) returns the value shifted to the right by the specified number of bits.

String functions:

- **COMPARE**(string1, string2) compares two strings, returning -1 when string1 is 'less than' string2, 0 when the strings are identical, and 1 when string1 is 'greater than' string2.
- **FIND**(string1, string2) returns the position in string1 where string2 is first found. If string2 is not found, the return is 0.
- **LEN**(string) returns the length of the specified string.
- **STRING_TO_INT**(string) converts the specified string to an integer.

Timer functions:

- **T_DONE**(timer_id) returns True when the specified timer has expired
- **T_PREVAL**(timer_id) returns the specified timer's preset value
- **T_VALUE**(timer_id) returns the specified timer's current value

The **Date/Time** functions are all used to format the given *seconds* parameter, as counted from January 1, 1970. To format the current time, you must first get the time using the [Date/Time Get command](#).

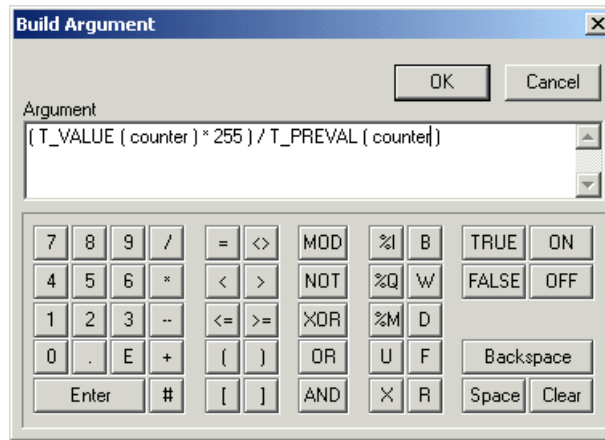
- To specify a string, insert the string's tag name in the function's parameter list.
- To specify a timer, insert the timer's tag name as the timer_id in the function's parameter list [T_DONE(timer tagname)].
- To specify the number of seconds since January 1, 1970, obtain the current system time using the Date/Time Get process block.

Tags

A tag references an item in the PointeControl database. When a device input scan or chart assignment statement assigns a new value to a tag, the entry in the database updates. When an expression references a tag, the runtime system retrieves the tag's current value from the database and uses the value in the expression.

Expression Syntax

In many instances, expressions can replace a single value or tag reference. Enter expressions as free form text; they may be arithmetic or logical in nature.



All expressions produce a numeric result—either the actual result of the arithmetic calculation, or the true/false result of a logical expression, where true is 1 and false is 0.

An expression can include constants, tags, and functions and follows the syntax:

expression operator expression

The following table summarizes the available operators, listed in order of precedence.

EXPRESSION TYPE	OPERATORS
unary	-a (negation)
multiplicative	a * b (multiplication) a / b (division) a MOD b (modulus)
additive	a + b (addition) a - b (subtraction)

EXPRESSION TYPE	OPERATORS
relational	a < b (less than) a <= b (less than or equal to) a > b (greater than) a >= b (greater than or equal to) a = b (equal to) a <> b (not equal to)
bitwise	a AND b (bitwise AND) a OR b (bitwise inclusive OR) a XOR b (bitwise exclusive OR)
logical	NOT a (logical NOT)

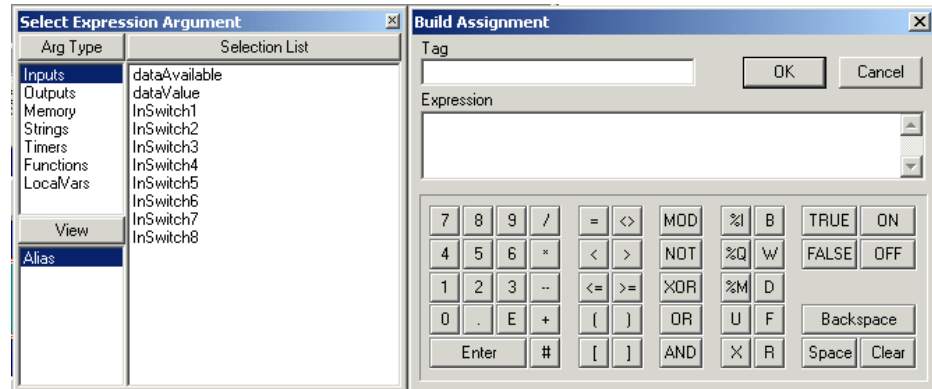
Calling Subcharts

The Build Argument dialog also appears when you define arguments to pass to a called subchart. This dialog includes one text entry field, Argument. You can enter math and Boolean logic in the Argument field, but you must enclose Boolean expressions in brackets. You cannot compare values (greater than or less than), and you can only use the AND Boolean operator.

Other Build Dialogs

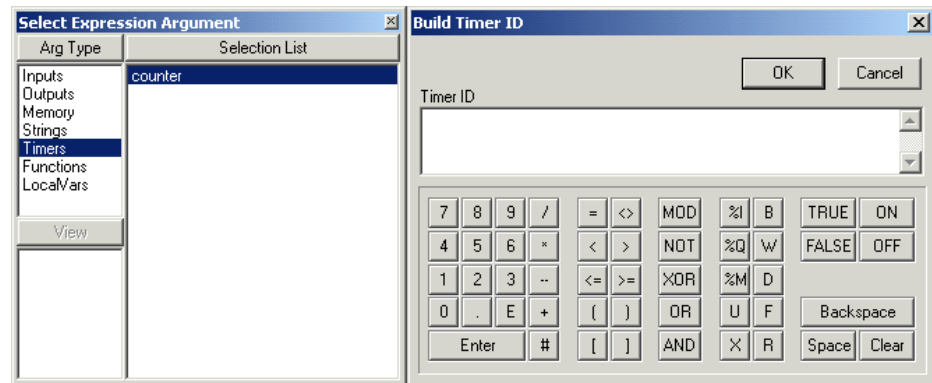
Certain block properties present customized “build” dialogs, other than the standard Build Expression dialog. These other dialogs are described below.

Build Assignment



The Build Assignment dialog appears when you define an **Assign command** in a Process block. A statement assigns the specified Expression value to the specified Tag. You can only enter one tag reference in the Tag field. You can enter math and Boolean logic in the Expression field, but you must enclose Boolean expressions in brackets. You cannot compare values (greater than or less than), and you can only use the AND Boolean operator.

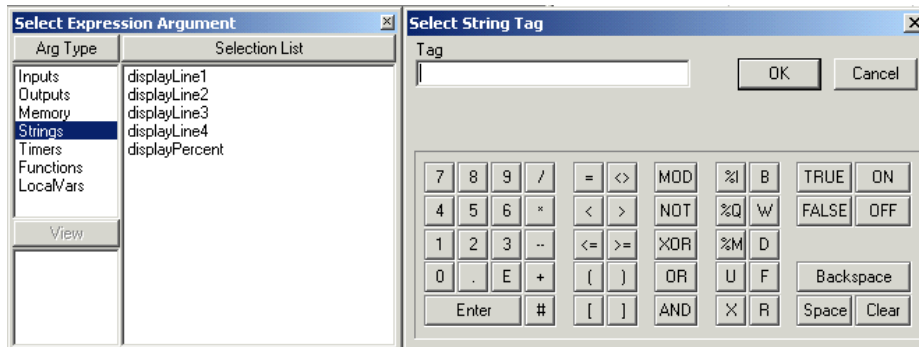
Build Timer ID



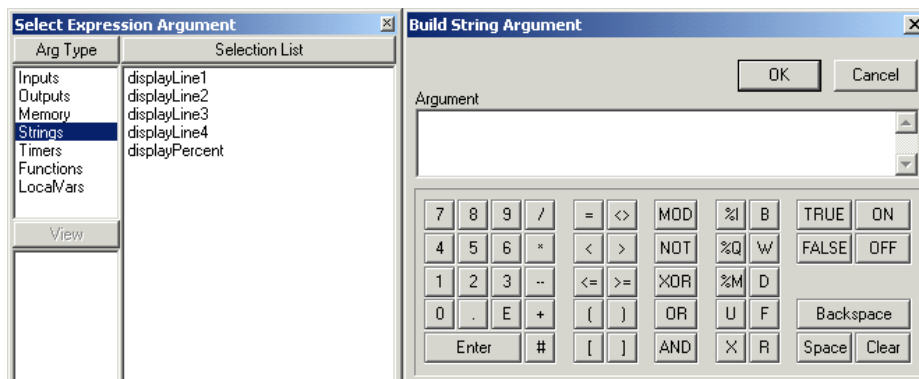
The Build Timer ID dialog appears when you define the properties for **Timer commands**. The dialog contains a single text box to enter the Timer ID, which can be selected from the list on the left.

Select String Tag and Build String Argument

When you select a **String command** in a Process block, a list of parameter types for that command appears. The first parameter, Destination String, displays a String Tag Select dialog when selected. You can enter only one tag reference as the string tag field.



The second and following parameters that appear depend on the selected string command. You may enter a Source String, Number of Chars, Start Position, String to Insert, or Replacement String. When you select any of these parameters, the Build String Argument dialog box appears so you can define the remaining arguments.

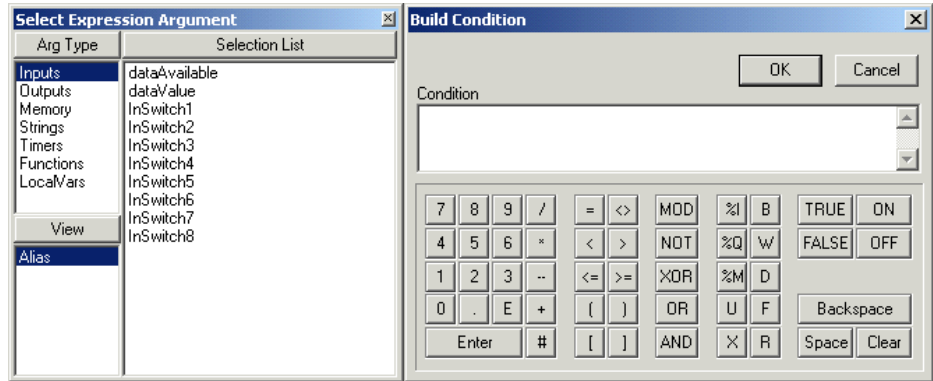


You can enter math and Boolean logic for the string command argument, but you must enclose Boolean expressions in brackets. You cannot compare values (greater than or less than), and you can only use the AND Boolean operator.

Build Condition

The following dialog boxes appear, depending on the Condition Type you select for a decision block of If/Then, Repeat/Until, or While Loop:

- For a Condition Type of **Expression**, the Build Condition dialog appears:



This dialog includes one text entry field, Condition, which must include some relational operator (e.g. =, <, >, ≠) to be evaluated. You can enter math and Boolean expressions in the Condition field, but you must enclose Boolean expressions in brackets.

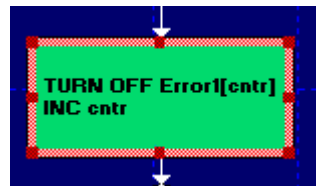
If the Condition evaluates as TRUE when the chart is scanned, then the true branch of the decision block will be followed. If the Condition evaluates as FALSE, then the false branch will be followed.

- For a Condition Type of **Diag Fault Bit Test** or **Error Status Bit Test**, a regular Select Tag dialog appears for you to enter the Source Tag.

5.6.5 Moving, resizing, and deleting blocks in a Flow Chart

Once a block or group of blocks has been placed in a flow chart, you can move, copy, resize, or delete it as needed without breaking the chart’s flow. The flowlines to and from the blocks are automatically redrawn to accommodate whatever changes you make.

- To select a block or group of blocks: click the **Select Tool** in the **Flow Chart toolbar** and then click anywhere inside a block. The selected block becomes dark green and red handles appear around it. To select multiple blocks, hold down the <shift> key while selecting blocks. All selected blocks become green.



NOTE: If you select a decision block (If/Then, Repeat/Until, or While/Do), then all of the blocks contained within the decision block's loop are also selected. These contained blocks are marked in yellow rather than green.

Moving a selected block

To move a selected block or group of blocks, click on it and drag it to a new location in the chart. You must place the moved blocks onto a flowline. The editor automatically inserts the block between the blocks connected by the flowline. Moved blocks carry all of their properties with them.

Copying a selected block

These editing features allow you to move and duplicate blocks. Cut and copy perform similar functions, but cut removes the original block and copy leaves the original in place. When you paste a block, the editor places it after the currently selected block. To copy a block:

1. With the **Select** tool, select a block to copy or cut.
2. To copy, click the **Copy** button, or choose **Copy** from the **Edit** menu.
3. To cut, click the **Cut** button, or choose **Cut** from the **Edit** menu.
4. Select a block after which the cut or copied block is to be pasted.
5. Click the **Paste** button, or choose **Paste** from the **Edit** menu. The block from the clipboard appears after the currently selected block.

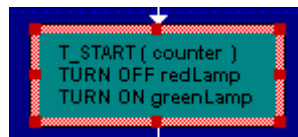
Resizing a selected block

After you select a single block, you can change the size of that block with the block handles. To change a block's size:

1. Pass the arrow tool over one of the handles until the tool becomes a double arrow (pointing in the directions you can resize with that handle).



2. Click the handle and drag the block outline to the size you want. You should only need to resize a block to accommodate a large label or name.



NOTE: You cannot manually resize multiple selected blocks. However, you can resize all of the blocks in the chart by using the **Size to Content** tool.

Deleting a selected block

There are many different ways to delete a selected block or group of blocks:

- Click either the **Cut Object** tool or the **Delete Object** tool on the **framework toolbar**. Cutting a block places it on the clipboard. Deleting a block does not place it on the clipboard.
- Choose either **Cut** or **Delete** from the Edit menu. Cutting a block places it on the clipboard. Deleting a block does not place it on the clipboard.
- Press the key on your keyboard, click on the Delete Object tool, or select **Delete** from the **Edit** menu.

Cutting a block places the block on the Windows clipboard. Deleting an element does not place the element on the clipboard.

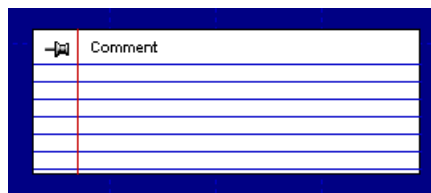
5.6.6 Adding comments to a Flow Chart

Enter comments and tack them to your Flow Charts like a note. Unlike the other block types, comments do not snap to the chart's grid. Place the comments anywhere on the chart. Comments can display any text, like information about the chart or its blocks, but comments do not affect program flow.

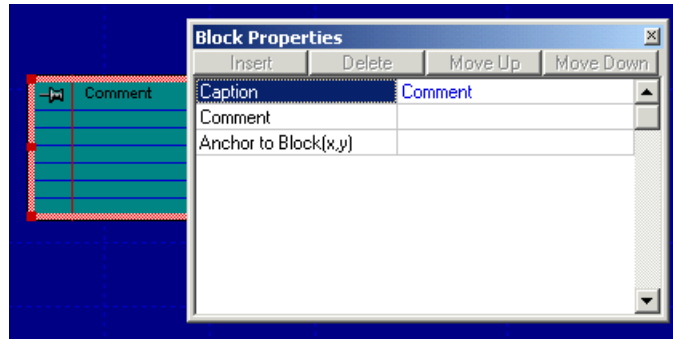
1. Click the **Comment** tool on the **Flow Chart toolbar**.



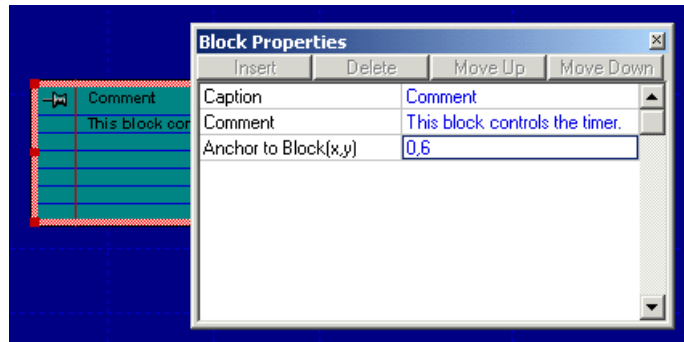
2. Point and click on the chart to begin placing the comment.
3. Drag the box until it reaches the size you want, then release the mouse button. The Comment block is now available for editing.



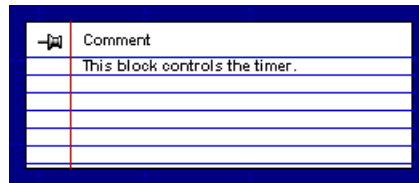
4. Double-click on the Comment block to open its Block Properties window.



5. Complete the block properties for the selected comment.



6. Close the Block Properties window.



If your comments should run larger than the limits of the block, you can resize the block. Text automatically reflows according to the size and shape of the comment block.

5.6.7 Logging changes in a Flow Chart

The View Change History button displays the change history of a Flow Chart if the audit log processing has been turned on for the selected chart. If audit log processing has not been turned on for the chart, then an error dialog appears that gives you the option of turning on the audit log processing. The program logs any changes made after the processing has been turned on in the change history for the chart.

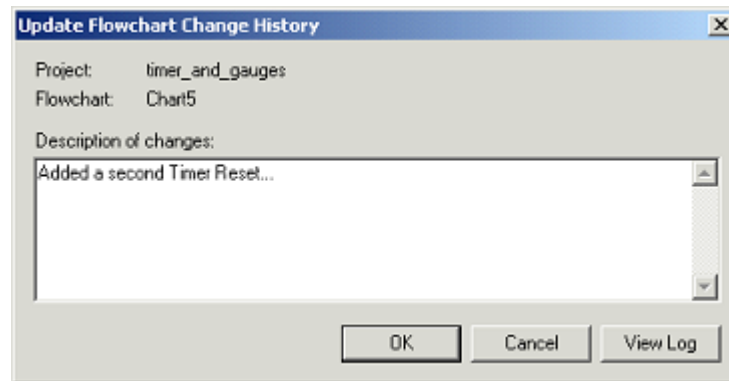
To enable audit log processing for a Flow Chart, click the **View Change History** button on the chart's toolbar:



A notice appears that gives you the option of enabling audit log processing.

NOTE: Once this feature is enabled, it cannot be disabled.

Once enabled, you are required to enter a comment on changes made to the chart each time you close the Flow Chart editor window (and save changes). The Update Flowchart Change History box appears, allowing you to enter your comments.



You can also check the current change history by clicking the **View Log** button.

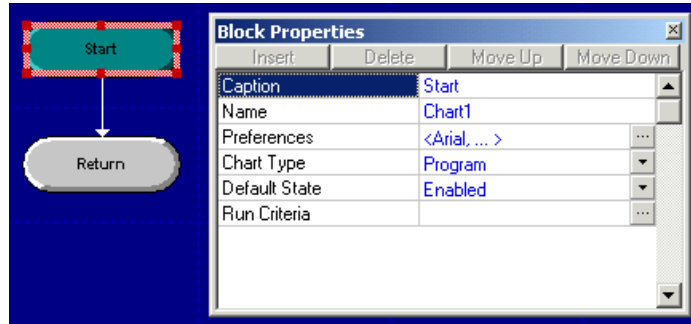
After you have enabled audit log processing for a Flow Chart, the Update Flowchart Change History box appears whenever you click the **View Change History** button on the toolbar.

5.6.8 Making a Flow Chart a reusable Subchart

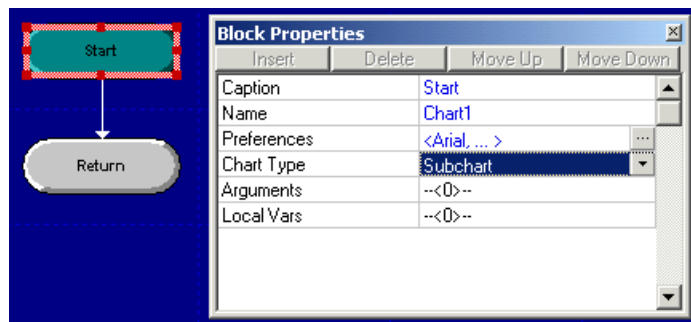
Create Subcharts to reuse common Flow Chart programming. This reuse speeds development and testing, since you only create and debug a chart once. When you create a Subchart, you add arguments specific to the chart. Another Flow Chart can then call the Subchart and assign values to these chart-specific tags.

To make a Flow Chart a reusable Subchart:

1. Double-click on the chart's **Start block** to open its Block Properties window.

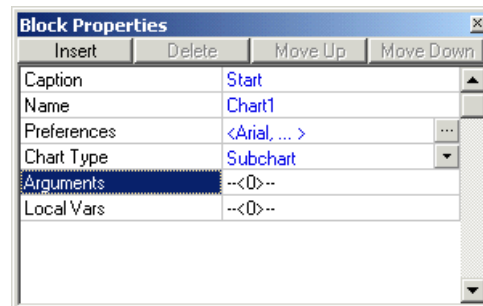


2. Click the **Chart Type** property and select **Subchart** from the drop-down menu. The Arguments and Local Vars properties are added to the window.

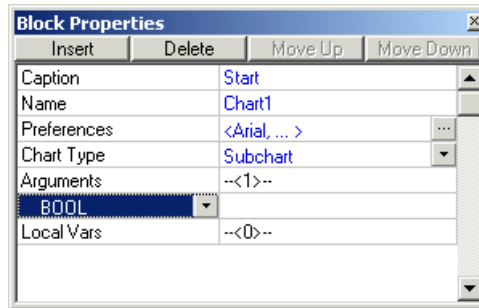


3. Define the arguments and local variables to be used by the Subchart:
 - a. Select **Arguments** or **Local Vars**. Arguments are values and variables passed from the calling Flow Chart to the Subchart. Local Vars are variables defined and used only within the Subchart, completely independent of Logic Memory.

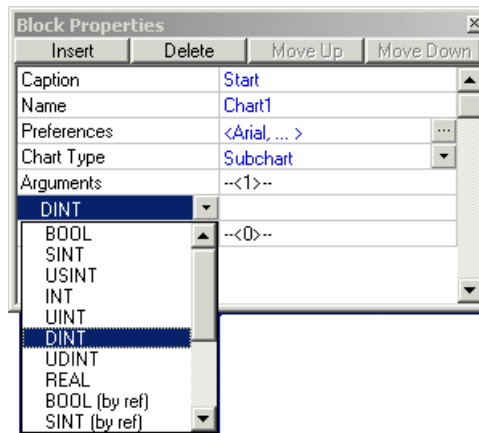
In this example, **Arguments** is selected:



- b. Click **Insert** to add a new argument.

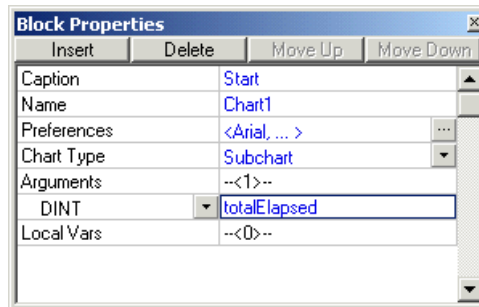


- c. From the tag type drop-down list, select the argument type. In this example, **DINT** (32 Bit Signed) is selected:



For more information on configuring arguments, see "A Note on Arguments" below.

- d. In the variable's text entry box (right column), enter a name to describe the tag. In this example, "totalElapsed" is entered:



- e. Repeat steps a through d for all of the arguments and local variables to be used by the Subchart.

4. Close the Block Properties window and save the Subchart. It can now be called by any other Flow Chart.

A Note on Arguments

When you define an argument in the Subchart's **Start block**, you are merely setting it up to receive some value or variable that will be passed from the calling Flow Chart. The actual value of that argument is defined in the Flow Chart's **Subchart block**.

Also, arguments can be passed either "by value" or "by reference":

- An **Argument by value** is the calculated value of the argument's **logical expression** at the moment the Subchart is called. The value is then operated on locally within the Subchart – as if it was a literal – without affecting the tag or expression from which the value was calculated.
- An **Argument by reference** is a pointer to a tag in Logic Memory. Operations on the pointer within the Subchart read from and write to the tag as if it was referenced directly by the Subchart. The pointer is used so that a different tag can be referenced each time the Subchart is called. It all depends on what tag reference is passed by the calling Flow Chart.

If the Subchart needs to read from or write to the same Logic Memory tag every time it is called, then the tag can be directly referenced the same way it is in a regular Flow Chart.

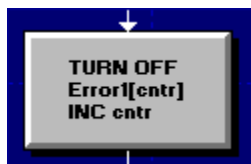
Make sure you select the right variable type – by value or by reference – from the type menu (step 4c above).

Calling the Subchart

For a complete description of how to call a Subchart from within a Flow Chart, see "**Subchart Block**" on page 161.

5.7 Types of Flow Chart Blocks

5.7.1 Process Block

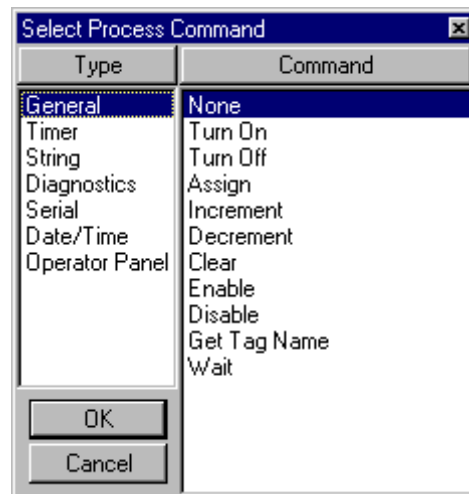


Process blocks define commands in your control program logic. You can add an unlimited number of the 41 available commands to a single Process block. Upon encountering a Process block, the program executes the commands in their order of appearance, from top to bottom, in the Block Properties window.

PROPERTY	WHAT YOU ENTER
Caption	User label for the block.
Command	<p>You can choose commands from the following types:</p> <ul style="list-style-type: none"> ▪ General Commands ▪ Timer Commands ▪ String Commands ▪ Diagnostics Commands ▪ Serial Commands ▪ Date/Time Commands ▪ Operator Panel Commands <p>Most commands require you to configure additional parameters. To add and remove additional commands, use the Insert and Delete buttons in the Block Properties window. To change the order of the commands, use the Move Up and Move Down buttons.</p>

Double-clicking on an empty Command property opens a Select Process Command dialog, which can be used to select a specific command and add it to the Process block.

First, select a command type from the Type list on the left. When a command type is selected, the commands available in that type appear in the Command list on the right. From there, select a specific command from the Command list and click **OK**. The selected command and its configurable parameters will be shown in the Process block's Block Properties window.



A complete description for each command – including configuration details – can be found in Appendix B, “Flow Chart Command Reference.” Page links are provided for each command type. **General Commands** (at right) can be found starting on page 298. The rest of the command types are shown below.

Type	Command
General	Timer Start
Timer	Timer Stop
String	Timer Reset
Diagnostics	Timer Preset
Serial	
Date/Time	
Operator Panel	

Timer Commands [page 303]

Type	Command
General	String Copy
Timer	String Concat
String	String Left
Diagnostics	String Right
Serial	String Mid
Date/Time	String Insert
Operator Panel	String Delete
	String Replace
	String Format Integer

String Commands [page 306]

Type	Command
General	Diag Get Tag Status
Timer	Diag Set Tag Status
String	Diag Clear Tag Status
Diagnostics	
Serial	
Date/Time	
Operator Panel	

Diagnostics Commands [page 316]

Type	Command
General	Serial Configure Port
Timer	Serial Enable Port
String	Serial Disable Port
Diagnostics	Serial Read Byte
Serial	Serial Write Byte
Date/Time	Serial Read MultiBytes
Operator Panel	Serial Write MultiBytes
	Serial Get Comm Errors

Serial Commands [page 321]

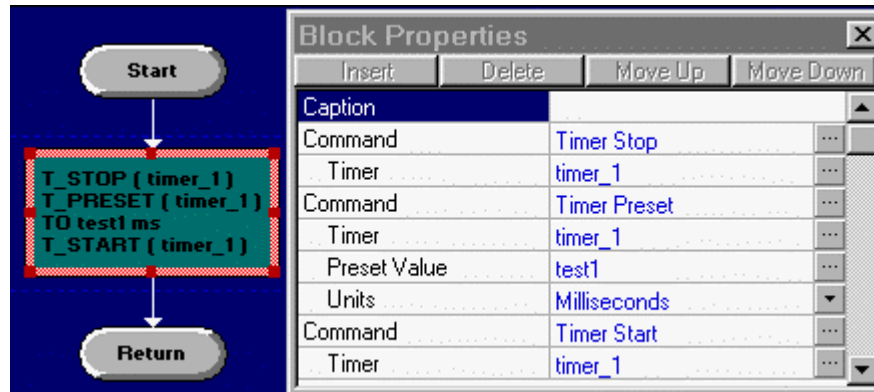
Type	Command
General	Date/Time Get
Timer	Date/Time Format
String	Get Elapsed Time
Diagnostics	
Serial	
Date/Time	
Operator Panel	

Date/Time Commands [page 330]

Type	Command
General	Keypad Data Entry
Timer	Arrow Adjust Data Entry
String	Button On
Diagnostics	Button Off
Serial	
Date/Time	
Operator Panel	

Operator Panel Commands [p. 333]

Process Block Example



Program flow enters the process block at the top and executes the commands as they are listed, from top to bottom.

- The command, Timer Stop, freezes the specified timer, timer1.
- The command, Timer Preset, sets the specified timer (loads Preset), timer1, to the value contained in the specified tag, test1.
- The command, Timer Start, restarts the specified timer, timer1.
- After all of the block's commands execute, program flow continues through the outgoing arrow to the next block.

5.7.2 Terminator Block

Terminator blocks define the start and exit points of a Flow Chart. Each chart has one Start block and at least one Return block. Start blocks for **subcharts** contain important information about chart-specific tags (see below).

Start Block Properties:

PROPERTY	WHAT YOU ENTER
Caption	User label for the block.
Name	Name of the Flow Chart, as it appears in the Project Workspace pane.
Preferences	<p>Display the Flowchart Preferences window, in which you can set certain appearance and behavior options for the chart:</p> <ul style="list-style-type: none"> ▪ In the Font pane, click the Select button to change the font, font size, font style, or font script. The default font is Small Fonts. ▪ In the Block Text Margins pane, use the controls to adjust the Left/right and Top/bottom margins. The default is 6 points. ▪ In the Size to Content Preferences pane, use the sliding scales to change the starting width of the Diamonds (decision blocks) and Rectangles (process blocks) in your chart. Also, use the control to adjust the Horizontal Snap Size (the incremental snap-to width; the higher the value, the wider the block). ▪ In the Options, select the check box if you want to: <ul style="list-style-type: none"> ▪ Always YIELD in loops (see note below) ▪ Always size to content ▪ Display block shadow ▪ Freeze selected block in detail view ▪ Check run criteria on all YIELDS (see note below) <p>Save or restore default selections by clicking the Save Defaults or Restore Defaults buttons, respectively.</p>
Chart Type	<p>Program (executable flowchart) OR Subchart (callable subchart)</p> <p>For more information on subcharts, see “Making a Flow Chart a reusable subchart” on page Error! Bookmark not defined.</p>
Default State (Program only)	<p>Enabled OR Disabled</p> <p>A disabled state inhibits the execution of the Flow Chart. You may find this useful for developing charts for future functionality in your project, or if some machinery is undergoing maintenance and should not run.</p>

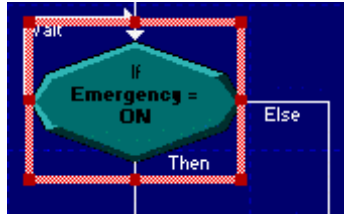
PROPERTY	WHAT YOU ENTER
Run Criteria (Program only)	Presents Build Condition dialog to specify a condition that must be true for the program chart to execute. Run criteria are always evaluated on start of execution at the beginning of a chart. You can also elect to evaluate run criteria each time a chart returns from a yield (for example, inside a loop or wait operation) by selecting the option in the Flowchart Preferences dialog box.
Arguments (Subchart only)	By Value (default): A tag's value is passed to the subchart; as such, the tag cannot be modified within the subchart. Literal constants, i.e., numbers, may be passed by value. By Reference: A reference to the tag is passed to the subchart. Since the tag's reference provides direct access to the PointeControl database, the tag's value can be referenced or modified within the subchart through the tag reference. A string may also be passed by reference.
Local Vars (Subchart only)	The number of chart-specific local variables. A listing of the defined chart-specific outputs appears below this property.

NOTE: A loop (Repeat/Until or While/Do) that is set to YIELD will allow the rest of the charts in the project's Chart List to scan while the loop is running. If the loop is not set to YIELD, then it will continuously check its run condition without interruption, effectively putting all other charts on hold. For more information, see "Repeat/Until Loop" on page 158 and "While/Do Loop" on page 159.

Return/Exit Block Properties:

PROPERTY	WHAT YOU ENTER
Caption	User label for the block.
Terminate Type	Return OR Exit For program charts, Return restarts the chart from the Start block; for subcharts, Return directs program flow to the calling chart. Exit directs program flow out of a loop and continues execution at the block immediately following the loop.

5.7.3 Condition (If/Then/Else) Block

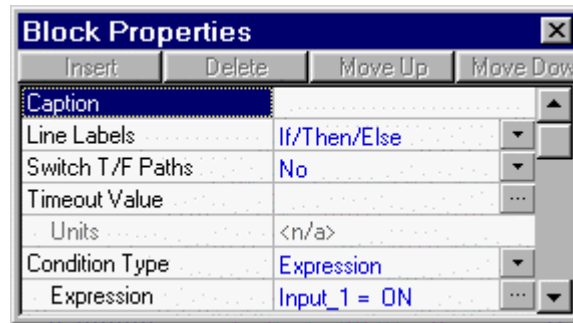


A Condition block contains a Yes or No type question that directs program flow in one of two directions. When the specified condition evaluates to a non-zero value, program flow proceeds down the true branch.

By default, the true branch is labeled “Then” and the false branch is labeled “Else,” to correspond to the if-then-else programming construct. You can change these labels to yes and no or on and off to correlate to the test condition.

PROPERTY	WHAT YOU ENTER
Caption	User label for the block.
Line Labels	If/Then/Else, Is/Yes/No, or Is/On/Off as the labels for the block and outgoing flowlines.
Switch T/F Paths	Yes or No. Yes switches the direction of the true/false paths. When you enable this option, the true path proceeds to the right, and the false path proceeds down. No leaves true/false paths in the normal orientation (true is down, false is to the right).
Timeout Value	Timeout value to wait for the condition to be true before proceeding down the false path. Only the current chart remains suspended waiting for this timeout to occur—all other charts continue to run normally.
Units	Units of the Timeout Value.
Condition Type	Type of Condition: <ul style="list-style-type: none"> ▪ Expression – Build a conditional expression using the Build Condition dialog. ▪ Diag Fault Bit Test – Select an Input, Memory, or Output tag and test to see if its Diag Fault Bit is set or clear. If the condition evaluates true, then the Then/Yes/On line is followed. If the condition evaluates false, then the Else/No/Off line is followed.

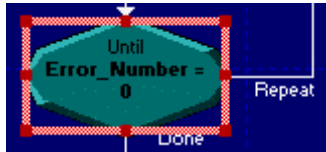
Condition Block Example



This decision block determines if Input_1 is ON or OFF. If Input_1 is ON, program flow continues through the Flowline labeled Then. If Input_1 is OFF, flow continues through the flowline labeled Else.

- The LineLabels property specifies a Then label for the flowline followed from a true or on condition and an Else label for the flowline followed from a false or off condition.
- The Switch T/F Paths property is No, meaning the default true/false path is taken. A selection of Yes switches the true/false path of the flow.
- The Timeout Value property is not selected. When you select a tag, the value adds a wait period to the block. Program flow does not follow the Else line, even if the condition is false, until the Timeout expires and the condition still evaluates to false. Any time the condition evaluates to True, regardless of the Timeout, program flow follows the Then line.
- The Units property is associated with the Timeout Value. If you specify a Timeout Value, you can select the units for counting time in milliseconds, seconds, or minutes.
- The Condition Type property is selected as an Expression, and the next property defines the block’s tested expression. In this condition, the tag, Input_1, can be ON or OFF.

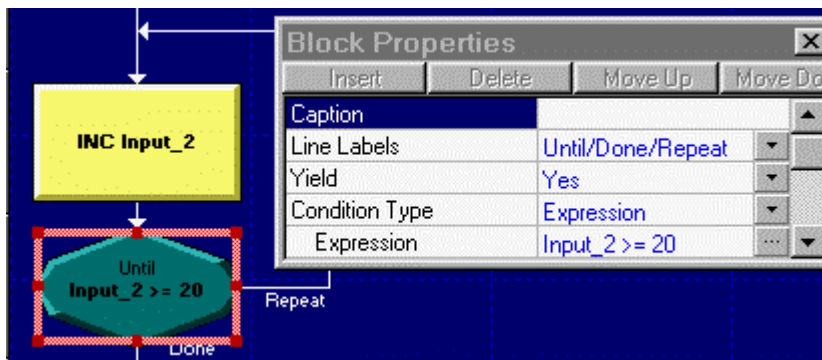
5.7.4 Repeat/Until Loop Block



The Repeat/Until Loop block represents a repetitive process, where the block checks the condition at the bottom of the loop, ensuring that the chart executes the sequence of instructions at least once. If the condition at the bottom of the loop is true, program flow continues at the next block in the chart. If the condition is false, control returns to the top of the loop.

PROPERTY	WHAT YOU ENTER
Caption	User label for the block.
Line Labels	Until/Done/Repeat or Until/On/Off as the labels for the block and outgoing flowlines.
Yield	Yes or No to specify whether, when the condition tests false, program flow should immediately return to top of the loop or yield to the next listed Flow Chart. Use this option with caution, since it essentially disables all other charts while the program executes the loop. A block that yields (Yes) runs all other charts in your Chart List , then returns to the top of the loop that yielded and retests the block's condition. A block that does not yield (No) continues to check the condition, uninterrupted.
Condition Type	Type of Condition: <ul style="list-style-type: none"> ▪ Expression – Build a conditional expression using the Build Condition dialog. ▪ Diag Fault Bit Test – Select an Input, Memory, or Output tag and test to see if its Diag Fault Bit is set or clear. If the condition evaluates true, then the Done/On line is followed. If the condition evaluates false, then the Repeat/Off line is followed.

Repeat/Until Loop Block Example

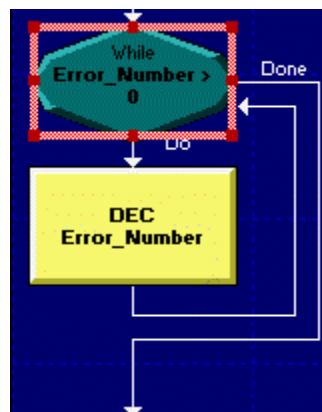


Before reaching this decision block, the program increments the value of Input_2. The decision block then determines if Input_2 is greater than or equal to 20. If

yes, program flow continues through the Flowline labeled Done. If Input_2 is less than 20, flow yields to the next chart, then returns to the flowline Repeat to again increment and check Input_2 until the required count is reached.

- The LineLabels property specifies a Done label for the flowline followed from a condition of Input_2 being greater than or equal to 20 and a Repeat label for the flowline followed from a condition of Input_2 being less than 20.
- The Yield property specifies the program flow must continue to the next chart if the condition is not met and return to increment and check the value again until the condition is met.
- The Condition Type property selects an Expression.
- The next property defines the block’s tested expression. In this condition, the tag, Input_2, can be any number greater than or equal to 20.

5.7.5 While/Do Loop Block

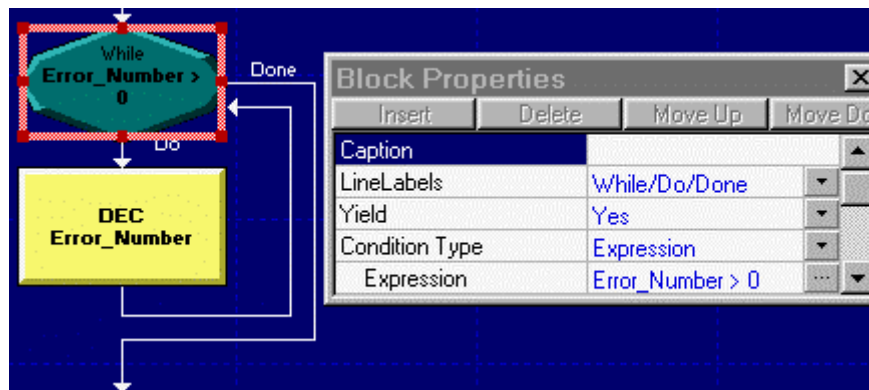


The While/Do Loop block represents a repetitive process, where the block evaluates a condition at the top of the loop until the condition evaluates to true. If the condition is true, the blocks in the loop are executed; if the condition is false, program flow continues at the block immediately following the loop.

PROPERTY	WHAT YOU ENTER
Caption	User label for the block.
Line Labels	While/Do/Done or While/On/Off as the labels for the block and outgoing flowlines.
Yield	Yes or No to specify whether, when the condition tests false, program flow should immediately return to top of the loop or yield to the next listed chart. Use this option with caution, since it essentially disables all other charts while the program executes the loop. A block that yields (Yes) runs all other charts in your Chart List after executing the blocks in the loop, then returns to the top of the loop that yielded and retests the block’s condition. A block that does not yield (No) continues to check the condition, uninterrupted.

PROPERTY	WHAT YOU ENTER
Condition Type	Type of Condition: <ul style="list-style-type: none"> ▪ Expression – Build a conditional expression using the Build Condition dialog. ▪ Diag Fault Bit Test – Select an Input, Memory, or Output tag and test to see if its Diag Fault Bit is set or clear. If the condition evaluates true, then the Do/On line is followed. If the condition evaluates false, then the Done/Off line is followed.

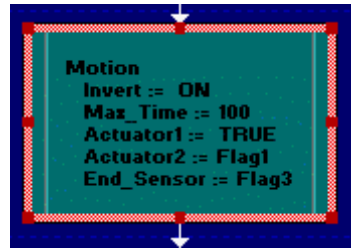
While/Do Loop Block Example



The decision block determines if Error_Number is greater than 0. While the answer is yes, program flow yields to the next chart, returns to the flowline labeled Do, and decrements the value of Error_Number before returning to the decision block to check the value. If the value is greater than 0, flow continues to the remainder of the current chart through the flowline labeled Done.

- The LineLabels property specifies a While label that specifies the condition, a Do label for the flowline from a condition of Error_Number being greater than 0, and a Done label for the flowline a condition of Error_Number being less than or equal to 0.
- The Yield property specifies the program flow must continue to the next chart while the condition is met and return to decrement and check the value again until the condition is no longer met.
- The Condition Type property is selected as Expression.
- The next property defines the block’s tested expression. In this condition, the tag, Error_Number, can be any number greater than 0.

5.7.6 Subchart Block



A Subchart block makes a call to a **previously defined subchart**, analogous to a call to a subroutine in a programming language. When program flow reaches a Subchart block, the chart transfers program flow to the Start block in the called subchart. Program flow continues from that point until reaching a Return block. Control then returns to the calling (main) chart and proceeds with the block immediately following the original

Subchart block. You can nest Subchart calls; a main chart may call a subchart, which can in turn call another subchart, etc.

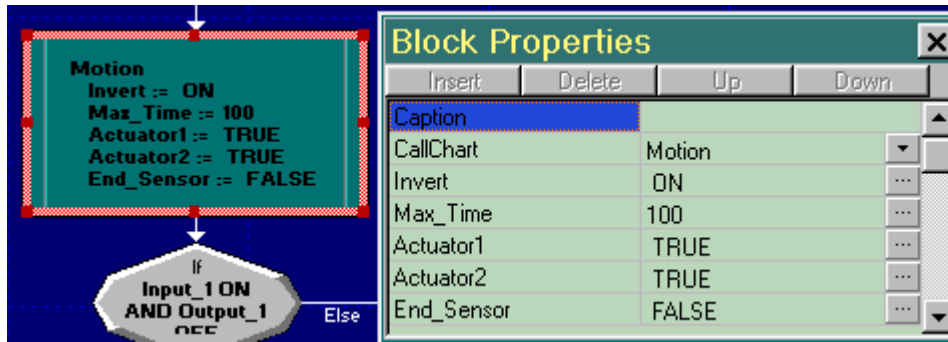
PROPERTY	WHAT YOU ENTER
Caption	User label for the block.
Call Chart	Name of the subchart to call.
Arguments	Arguments to pass.
Local Vars	Variables local to the subchart.

Subchart Arguments

When calling a function chart, you must assign tag references to the called chart's internal tags. To assign these values, pass arguments from the calling chart's subchart block to the called chart (specified by CallChart):

1. Add or select a Subchart block.
2. In the Block Properties box, select a chart from the CallChart drop-down list. After you select a chart, the properties appear in the Block Properties box.
3. Press the logic statement button for each property to build the arguments that are passed from the calling chart's subchart block to the called subchart. For internal references, you can assign a value or reference any tag.
4. Access the local variables that are to be used within the subchart. The local variables are not accessible from other charts.

Subchart Example



This subchart block calls the function chart, Motion. Program flow enters the block at the top and jumps to the Motion chart's start block. The subchart block passes each argument, listed in the Block Properties box (Invert becomes ON, Max_Time becomes 100, etc.) to that particular call of the Motion chart. Program flow continues through the Motion chart until reaching a return block. Flow returns to the calling chart and continues through the outgoing flowline of the subchart block.

5.8 Building and Editing Ladder Diagrams

A Ladder Diagram is a program component written using traditional Relay Ladder Logic. The ladder is made up of sequential rungs, and each rung is made up of sequential function blocks.

Ladders are executed from top to bottom, one rung at a time.

5.8.1 Creating a new Ladder Diagram

To create a new Ladder Diagram:

3. In the Project Workspace pane, select **Ladder Diagram**.
4. Click the **New Object** tool in the Framework Editor toolbar, or choose **New** from the **File** menu. A new Flow Chart with a default name (LadderN) will be added to the project hierarchy.

To open a Ladder Diagram for editing, simply double-click on it *or* select the diagram and click the **Open Object** tool in the Framework Editor toolbar. When you open a diagram, its editor window will appear in the Object Editor pane.

Saving a Flow Chart

All program objects are saved automatically whenever you close them. However, you can save a Ladder Diagram while it's open by clicking the **Save Object** or **Save All Objects** tool in the Framework Editor toolbar.

Deleting a Flow Chart

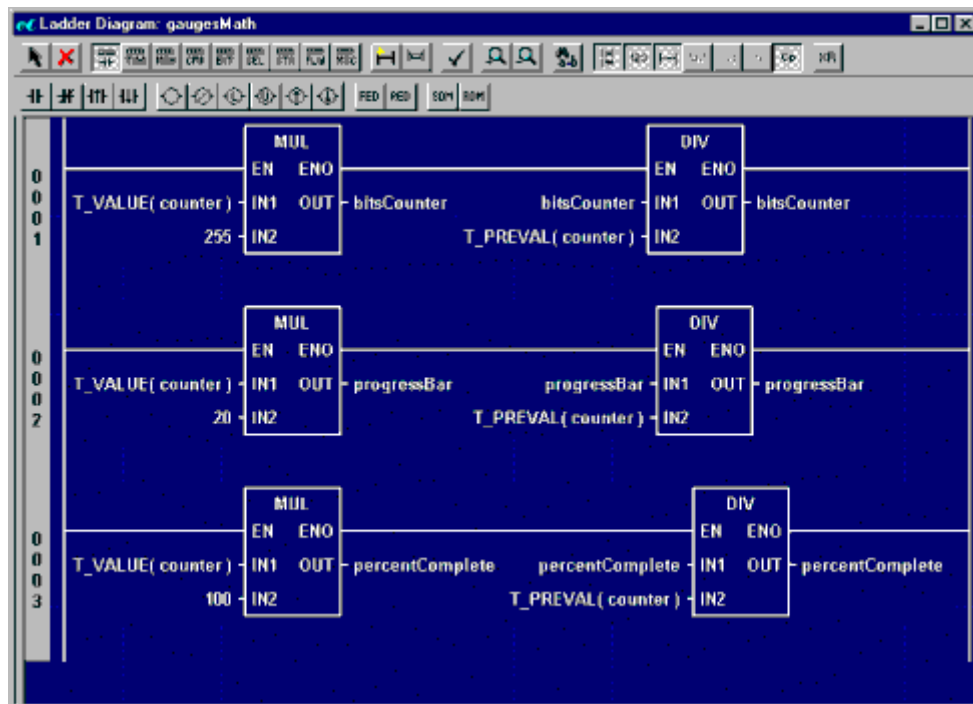
You can delete Ladder Diagram in the same place you open them: the Project Workspace pane. To delete a Ladder Diagram:

4. Close all open Flow Charts, Ladder Diagrams, and Logic Memory tables.
5. Select the desired Ladder Diagram in the Project Workspace pane. (You may need to expand the hierarchy to select it – double-clicking expands it, double-clicking again collapses it.)
6. Click the **Delete Object** tool in the Framework Editor toolbar, or choose **Delete** from the **Edit** menu.

NOTE: Before deleting a Ladder Diagram, PointeControl will ask for a confirmation.

5.8.2 Navigating the Ladder Diagram editor

As a resource of PointeControl, ladder diagrams appear in the resource tree of the PointeControl framework. Expanding the ladder diagrams entry in the resource tree shows the diagrams included with the project. Double clicking on one of the listed diagrams will activate the ladder editor and load the selected diagram. A new diagram can be defined by selecting the Ladder Diagrams entry in the resource tree and then clicking the New Object icon in the framework toolbar. Alternately, select File/New while the Ladder Diagrams entry is selected in the resource tree. The ladder editor window is shown below:

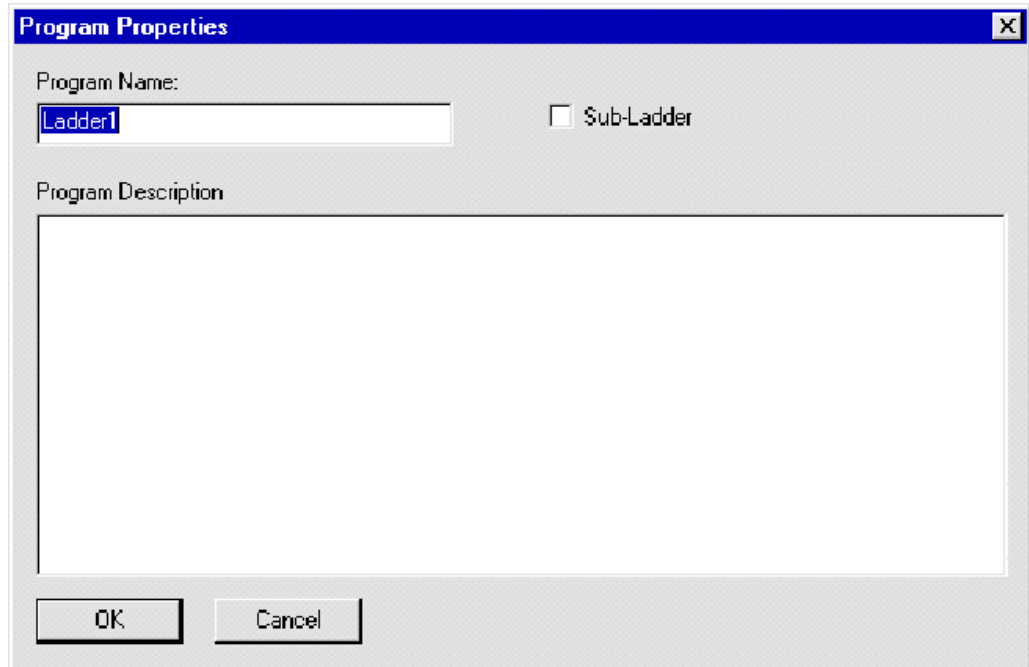


The window includes a title bar containing the name of the ladder diagram and a standard system menu bar. A toolbar, found under the title bar, contains a set of icons for each of the tools available to the ladder editor. A secondary toolbar, immediately under the main toolbar, provides the ladder objects available for placement in the workspace. This secondary toolbar changes with the selection of an object library tool from the main toolbar. The workspace, appearing under the toolbars, is the area in which the ladder diagram will be constructed. Scroll bars appear around the workspace as needed based on workspace contents and magnification level.

The workspace is the scratchpad into which ladder diagram blocks are placed. A ladder diagram consists of a series of ladder rungs that execute sequentially, from left to right, and top to bottom. Each rung consists of a left and right power rail and ladder objects, typically contacts and coils, as well as function blocks. Rungs may include branches and may also include jumps to other rungs or calls to other ladder diagrams. Objects are placed in the workspace using the tools contained on the toolbar and discussed in subsequent paragraphs.

The workspace area is shared with a cross-reference window and a code view window in which the structured text equivalent of the ladder diagram can be viewed. The Show Cross Reference tool (see subsequent paragraphs describing the editor tools) activates the cross-reference window. Dragging the splitter bar on the left side of the editor workspace to the right reveals the code view window. This is a view-only window – structured text cannot be edited.

Each ladder diagram has a set of properties: diagram name, type, and description. The name is limited to 30 characters (like Flow Charts) and will default to LadderX where X will be the next sequential index of all ladder diagrams for the project. The type can be either program (the default) or sub-ladder, allowing it to be called like a subroutine from another ladder diagram. The optional description can be text of essentially unlimited length describing the overall operation or purpose of the diagram. These properties can be defined when the diagram is first created or later, using the Program Properties dialog. This dialog is activated by a right click anywhere within the workspace (to activate a context menu) and selection of the Edit Program Properties entry. The dialog appears as:



Once the program properties are as desired select the OK button to save the values or select Cancel to leave the properties unchanged.

The Ladder Diagram toolbar



Access the Ladder Diagram programming tools from this bar. The toolbar allows you to select blocks, place a variety of block types, change zoom levels and special viewing mode, hide/show block labels, and check diagram integrity. To select a tool, simply click its button.

The tools are, from left to right:

Select Delete	These two buttons are used to select and delete elements in the Ladder Diagram. [page 168]
Relays and Coils Timer and Counter Math Comparison Logical and Bit Shift Selection String Flow Control Miscellaneous	These nine buttons are used to access the various libraries of Ladder Diagram function blocks. [page 167]
New Rung Branch Rung	These two buttons are used to add new rungs and rung branches to a Ladder Diagram. [page 166]
Validate Ladder	This button checks the structure and syntax of the Ladder Diagram. [page 181]
Zoom In Zoom Out	These two buttons adjust the magnification level at which the Ladder Diagram is displayed in the workspace. [page 177]
Replace Text	This button finds and replaces text in the Ladder Diagram. [page 177]
Show Rung Comments Show Rung Numbers Show Rung Logic Show Block Captions	These four buttons change what supplementary information is displayed in the Ladder Diagram workspace. [page 169]
Show Cross Refs	This button displays a list of all tags and variables used in the Ladder Diagram. [page 178]

5.8.3 Adding new rungs and branches to a Ladder Diagram



Rungs may be added to a ladder diagram via the New Rung tool. Selecting the New Rung button will add another rung to the current displayed ladder diagram. Rungs are inserted before a selected rung or added to the end of the diagram when no rung or block is selected, or when any block is selected. When a new rung is inserted or added in the wrong position, select the rung (click on the rung number) and drag it to the desired position, or delete it and try again.

Rungs may optionally be assigned a label to serve as the target for a GOTO block or a caption to document the diagram. Both attributes can be edited by a right-click on the rung number (to activate the context menu) and selecting the Edit Rung Properties entry.

Adding a branch to an existing rung



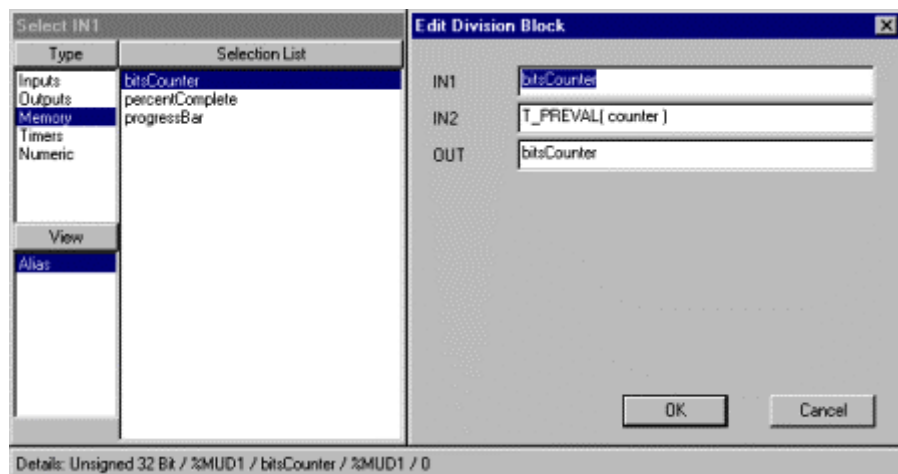
Branches may be added to rungs using the Branch Rung tool. This tool, when selected, allows you to click anywhere on a rung to select one endpoint for the branch. An arc then appears between the endpoint and the cursor position showing the path the branch will take. Now click on the rung to define the other branch endpoint. When a valid endpoint is specified the branch will be drawn. (Note the branch is not drawn as an arc once its endpoints are defined. Rather the branch corners are squared so the branch appears as a set of horizontal and vertical lines.) When an invalid branch endpoint is selected, an audible beep occurs and the arc remains visible so that a valid endpoint may be specified. To abort the branch definition, select another tool or hit the keyboard ESC key.

Branches cannot cross other branches, nor can they jump from one rung to another. Branches may nest within other branches. Once placed, the branch endpoints may be repositioned via drag-and-drop, subject to the previous restrictions.

5.8.4 Placing and configuring a Ladder Diagram block

Once placed on a rung within a ladder diagram, ladder function blocks must be configured, that is, each of the block's inputs and outputs must be assigned a tag from the project's database. A double click on the block, or a right click and selection of Edit Block Properties from the context menu, initiates the block configuration dialog.

The function block configuration dialog provides a convenient, easy-to-use way to provide tags for configuring ladder function blocks. Its functionality and appearance are similar to the configuration dialogs used in the chart editor. A representative example, showing the configuration of a division function, appears below:



The configuration dialog is actually two dialogs working together, one on the right, the configurator, providing values for each of the tags needed to configure a ladder function block, and one on the left, the browser, to assist with the tag selection. The configurator title bar shows the type of function block being

edited. As each of the configuration fields is selected the title bar of the browser updates to show the name of the field being edited. Use of the browser is optional – each of the configurator fields allows direct type-in. Validation is performed on each field and the configurator OK button is enabled only when all fields contain legal entries or blanks. (Blank entries are valid for some fields, for example the Timer Pulse preset and elapsed time fields. Checking for blanks in fields requiring a valid tag is performed at project build time.)

Use of the browser is straightforward. The Type listbox shows the valid database types for the selected configurator field. Types shown may be any or all of Inputs, Memory, Outputs, Strings, Timers, Numeric, or Literal. The inclusion of the Numeric and Literal types is for mnemonic purposes only – both types must be entered directly by the user. Literals must be enclosed within single quotes to be valid. Selection of a Type will force an update to the contents of the View and Selection List sections.

The View listbox shows the different views available for the selected database type. Inputs, Memory, and Outputs are viewed only by Alias. The available selections for Timers are T_DONE, T_PRESET, and T_VALUE when integer value tags are allowed in the selected configurator field. No View selections are shown when just the name of a timer is expected in the selected configurator field (as for example the Timer field for any of the Timer function blocks). No View selections are available for the String, Numeric, or Literal types.

The Selection List shows the available tags for the selected type and view. The list is filtered to exclude tags not appropriate for the selected configurator field. For example, when configuring fields for contacts or relays, only bit type tags of the selected type and view will be shown. Further filtering is available through use of the configurator field itself. Characters entered into the field are matched against the available tags and only those tags beginning with the entered characters will be shown: enter 'a' and all tags beginning with 'a' will be shown. Add the character 'r' and only tags beginning with 'ar' will be shown. Filtering is not case sensitive. A single click within the Selection List copies the selection to the active configurator field.

5.8.5 Moving, copying, and deleting elements in a Ladder Diagram



Elements (blocks, branches, rungs) are selected using the Select/Edit tool. Selecting an element highlights it and deselects/un-highlights any prior selection. Blocks are selected by clicking on the block. Branches are selected by clicking on the merge point. The branch origin is selected by clicking on the split point. Rungs are selected by clicking on the rung number. Once selected, elements may be dragged and dropped to other positions in the Ladder Diagram. Elements will drop only on legal drop points; if the drop point is not a legal position, then the element is not moved.

One exception to the drag-and-drop paradigm is the branch. Only the branch origin (split) and the branch termination (merge) may be dragged and dropped to new positions. To move or copy a branch to a new position, first Cut or Copy the selected branch. Next select another object around which the branch should link, then 'Paste' it in place. Should the branch origin and endpoint end up misplaced, simply drag and drop them to the desired position. Branches may not cross each other; they must always nest either inside or around other branches.

Deleting elements



Selected elements may be deleted using the Delete tool. Object deletion is also possible using the Edit menu or the keyboard DEL key. Be advised that element deletion is not undoable. Unlike cutting an element, the Delete action does not place a copy of the element in the clipboard.

5.8.6 Adding comments to a Ladder Diagram

Ladder Diagrams may be given optional comments or captions as an aid in documenting the logic flow.

Program Properties

Program properties include the Program Name (as it appears in the Project Workspace pane), the Program Description, and the **Sub-Ladder** option. To edit these properties, right-click anywhere in the Ladder Diagram and choose **Edit Program Properties** from the context menu.

Rung Properties

Rung properties include the Rung Label and Rung Comment. To edit these properties, right-click on the desired rung and choose **Edit Rung Properties** from the context menu.



The visibility of rung comments may be toggled using the **Show Rung Comments** tool in the Ladder Diagram **toolbar**. Rung comments are visible by default whenever the ladder editor is started.

Block Captions

Each function block in a Ladder Diagram can have its own Block Caption. To edit a block's caption, right-click on the desired block and choose **Edit Block Caption** from the context menu.



The visibility of block captions may be toggled using the **Show Captions** tool in the Ladder Diagram toolbar. Rung comments are visible by default whenever the ladder editor is started.



NOTE: Rung logic (the contacts, coils and function blocks comprising the ladder diagram) may be shown or hidden using the **Show Rung Logic** tool. Rung logic is visible by default whenever the ladder editor is started. This feature may be useful when comments are defined for each ladder rung. By hiding the rung logic, leaving just the comments visible, a text only description of the ladder diagram will be seen, allowing the viewer a quick overview of the purpose and functionality of the ladder diagram.

5.8.7 Making a Ladder Diagram a reusable Sub-Ladder

Create Sub-Ladders to reuse common Ladder Diagram programming. This reuse speeds development and testing, since you only create and debug a chart once.

To convert an existing Ladder Diagram into a reusable Sub-Ladder:

1. Select and open the desired Ladder Diagram from the **Project Workspace pane**.
2. Right-click anywhere in the ladder and choose **Edit Program Properties** from the pop-up menu. The Program Properties window will appear.
3. Click the **Sub-Ladder** checkbox.
4. Click **OK** to save your changes and close the window.

The program is now a reusable Sub-Ladder and can be called from another Ladder Diagram using the **CALL** and **RETN** blocks.

NOTE: Unlike Subcharts, Sub-Ladders cannot receive any local variables.

5.9 Types of Ladder Diagram Blocks

To make it easier to select from the 79 available Ladder Diagram blocks, they are grouped into nine block types or libraries. These libraries can be accessed by selecting the corresponding tools from the Ladder Diagram toolbar:



Clicking on a library toolbar button will make that library’s own toolbar appear just below the main Library Diagram toolbar. From there, you can select individual blocks and place them in your diagram.

TIP: A complete description for each block – including configuration details – can be found in Appendix C, “Ladder Diagram Block Reference.” Direct page links are provided below.




5.9.1 Relays and Coils



Selecting this library tool activates the Relays and Coils toolbar, which allows the placement of relay contacts and coil-type objects in your ladder diagram.

This library includes the following blocks:

TOOL	DESCRIPTION	FOUND ON...
	Normally Open Contact	Page 338
	Normally Closed Contact	Page 338
	Rising Edge Relay	Page 339
	Falling Edge Relay	Page 339
	Output Coil	Page 340
	Negated Output Coil	Page 341
	Latched Coil	Page 341
	Unlatched Coil	Page 342
	Rising Edge Coil	Page 343
	Falling Edge Coil	Page 343
	Falling Edge Detector	Page 344







TOOL	DESCRIPTION	FOUND ON...
	Rising Edge Detector	Page 345
	Set-Dominant Bistable	Page 346
	Reset-Dominant Bistable	Page 348

5.9.2 Timer and Counter Blocks



Clicking on this toolbar button activates the Timer and Counter Blocks toolbar, which allows the placement of timing and counting functions in your ladder diagram.

This library includes the following blocks:



TOOL	DESCRIPTION	FOUND ON...
	Timer, Pulse	Page 350
	Timer, ON Delay	Page 351
	Timer, OFF Delay	Page 352
	Counter, Up	Page 353
	Counter, Down	Page 355
	Counter, Up/Down	Page 356

















5.9.3 Math Blocks



Clicking on this toolbar button activates the Math Blocks toolbar, which allows the placement of mathematical functions in your ladder diagram. Activation of the Math Blocks toolbar terminates any active block insertion mode, forcing a selection from the Math Blocks library before insertion mode is restored.

This library includes the following blocks:

TOOL	DESCRIPTION	FOUND ON...
	Add	Page 359
	Subtract	Page 360


TOOL	DESCRIPTION	FOUND ON...
	Divide	Page 362
	Multiply	Page 363
	Square Root	Page 364
	Modulus	Page 366
	Sine	Page 367
	Cosine	Page 368
	Tangent	Page 369
	Arc Sine	Page 370
	Arc Cosine	Page 371
	Arc Tangent	Page 372
	Absolute Value	Page 373
	Logarithm	Page 374
	Natural Logarithm	Page 375
	Exponential	Page 376
	Natural Exponential	Page 378
	Expression	Page 379






5.9.4 Comparison Blocks



Clicking on this toolbar button activates the Comparison Blocks toolbar, which allows the placement of functions that perform a numerical comparison between two Logic Memory tags.

This library includes the following blocks:

TOOL	DESCRIPTION	FOUND ON...
	Greater Than	Page 381









TOOL	DESCRIPTION	FOUND ON...
	Greater Than or Equal to	Page 382
	Equal to	Page 383
	Not Equal to	Page 385
	Less Than or Equal to	Page 386
	Less Than	Page 387

5.9.5 Logical and Bit Shift Blocks



Clicking on this toolbar button activates the Logical and Bit Shift Blocks toolbar, which allows the placement of function block objects in the ladder diagram that perform Boolean and/or bit oriented functions.

This library includes the following blocks:





TOOL	DESCRIPTION	FOUND ON...
	And	Page 389
	Or	Page 391
	Exclusive Or	Page 393
	Not	Page 394
	Shift bits Left	Page 396
	Shift bits Right	Page 397
	Rotate bits Left	Page 399
	Rotate bits Right	Page 400

5.9.6 Selection Blocks



Clicking on this toolbar button activates the Selection Blocks toolbar, which allows the placement of selection or clamping functions in your ladder diagram.

This library includes the following blocks:












TOOL	DESCRIPTION	FOUND ON...
	Select minimum value	Page 402
	Select maximum value	Page 403
	Limit value	Page 405
	Select one of two values	Page 406

5.9.7 String Blocks



Clicking on this toolbar button activates the String Blocks toolbar, which allows the placement of string operations in your ladder diagram.

This library includes the following blocks:



TOOL	DESCRIPTION	FOUND ON...
	Set string	Page 408
	Find string length	Page 409
	Extract sub-string from Left	Page 410
	Extract sub-string from Right	Page 411
	Extract sub-string from Middle	Page 413
	Concatenate strings	Page 414
	Compare strings	Page 416
	Insert sub-string	Page 417
	Delete sub-string	Page 419
	Replace sub-string	Page 421
	Find sub-string	Page 423

5.9.8 Flow Control Blocks



Clicking on this toolbar button activates the Flow Control Blocks toolbar, which allows the placement of command objects that control execution flow of the ladder diagram. These objects alter the standard left to right, top to bottom execution of the diagram.

The library includes the following blocks:











TOOL	DESCRIPTION	FOUND ON...
	Call sub-ladder diagram	Page 425
	Return to main diagram	Page 426

5.9.9 Miscellaneous Blocks



Clicking on this toolbar button activates the Miscellaneous Blocks toolbar, which allows the placement of conversion and assignment functions in your ladder diagram.

The library includes the following blocks:

TOOL	DESCRIPTION	FOUND ON...
	Convert to Boolean	Page 427
	Convert to Integer	Page 428
	Convert to Float	Page 429
	Convert to String	Page 430
	Truncate	Page 432
	Integer to Character	Page 433
	Character to Integer	Page 434
	Integer to BCD	Page 435
	BCD to Integer	Page 436
	Move	Page 438

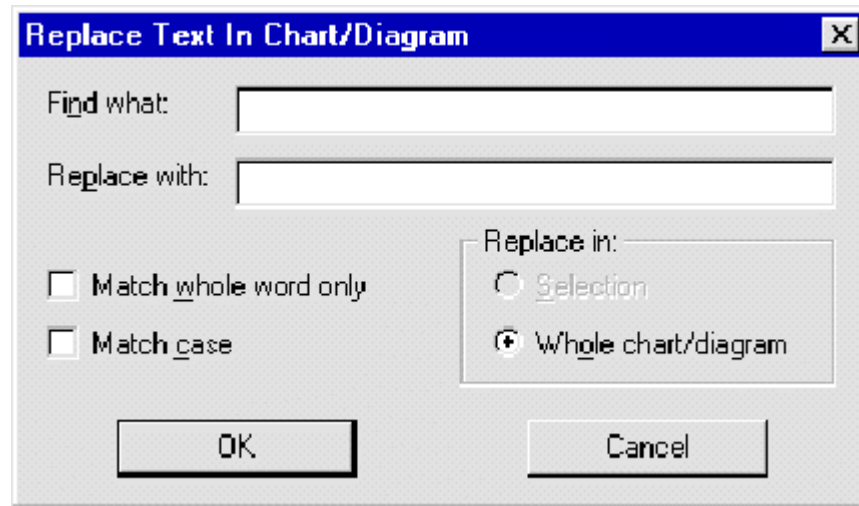
5.10 Other Framework Editor Tools

5.10.1 Finding and replacing text



Text strings within the active ladder diagram can be replaced with other strings using the Replace Text tool. This contrasts with the **Replace Text in Project** command that is available in the Edit menu, which replaces text throughout the project.

Selecting the tool activates the Replace Text dialog:



Using the dialog you specify the text string to be replaced and its replacement. Scans may be case-sensitive or case-insensitive. Scans may also be restricted to whole-word matches or not. This is especially useful when replacing tag names in a diagram and you need to replace all occurrences of the tag XIC1 with XIC2, but do not want to modify tags XIC11, XIC100, XIC1102, etc. The scan will default to replacing all occurrences of the search string within the whole diagram, unless there are objects selected within the diagram, in which case the scan defaults to the selected objects.

5.10.2 Zooming in and out on a chart



You can view the Flow Chart or Ladder Diagram editor workspace at several magnification levels using the **Zoom In** and **Zoom Out** buttons on the toolbar. Zoom In makes the diagram appear larger, while Zoom Out makes the diagram appear smaller, allowing you to see more of it within the workspace window. By default, the workspace is initially displayed at the maximum magnification level.

5.10.3 Viewing tag cross references



A cross-reference of all **Logic Memory** tags and variables used within the active Flow Chart or Ladder Diagram can be shown or hidden using the Show Cross-Reference tool. The cross-reference appears on the left side of the editor workspace and consists of a tree of all database alias tags. Each tag can be expanded, by clicking the open icon (cross within a square) or double clicking on the entry, to show the DRV, wire label and list of blocks using the tag. Each entry in the list includes the rung number and block type using the tag. A double click on the entry will select and show the function block. The cross-reference is hidden by default whenever the ladder editor is started.

5.11 Compling Your PointeControl Project

After you have developed your project, you can compile it into a finished program and download it to your Pointe Controller unit for execution.

To compile your PointeControl project for download:

1. Configure your Chart List.
2. Set your Scan Interval.
3. Build your runtime module.
4. Activate the PointeControl Monitor.

For more information on using PointeControl Monitor, see Chapter 6, "Downloading to the Controller," and Chapter 7, "Monitoring and Debugging."

NOTE: PointeControl does not support online changes. If you wish to modify your project *after* you have downloaded it to the Pointe Controller unit, you must make your changes in the PointeControl Framework and then recompile your project.

5.11.1 Configuring your project's Chart List

You can use the Chart List to specify which Flow Charts and Ladder Diagrams are to be compiled into the finished program and in what **scan order** they must be arranged.

NOTE: The Chart List does not list subcharts. Any subcharts called by charts in the Chart List are automatically included when the project is compiled.

To configure your project's Chart List:

1. From the **Project** menu, choose **Configure Chart List**. The **Chart List window** appears. All saved Flow Charts and Ladder Diagrams are listed under Available Charts on the right.
2. Under Available Charts, highlight the chart you want to add and click the <- button. The chart is moved from Available Charts on the right to the Chart List on the left.
3. Repeat step 2 for each chart you want to add to the Chart List.
4. To change the scan order of a chart in the Chart List, highlight the chart and use the **Up** and **Down** buttons to move it within the list. Repeat this step until all of the charts are in the correct scan order.
5. To remove a chart from the Chart List, highlight the chart and click the -> button. The chart is moved from the Chart List on the left to Available Charts on the right.

NOTE: This action only removes the chart from the project's Chart List. It does not delete the chart from the project. To delete a chart from the project, see [Deleting an object from your project](#).

6. When you are satisfied with the configuration of the Chart List, click **OK** to save your changes and close the window.

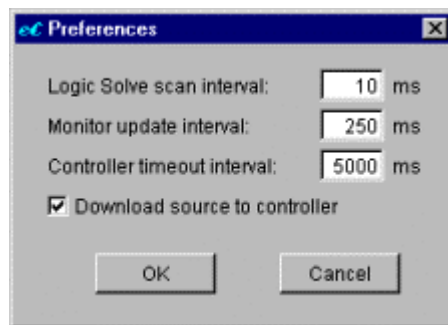
If you are doing a full compile of your project, then proceed to [Setting your project's scan interval](#).

5.11.2 Setting your project's scan interval

Your project's *scan interval* is the frequency at which the Pointe Controller unit *scans*, or resolves, the Flow Charts and Ladder Diagrams that make up your project. You can set the scan interval and other project preferences by using the Preferences window.

To set your project's scan interval:

1. From the **Edit** menu, choose **Preferences**. The Preferences window will appear.



2. Change the preferences as needed:
 - **Logic Solve scan interval** – The frequency at which the controller resolves the charts that make up your project. (Default: 10 msec)
 - **Monitor update interval** – The frequency at which the controller sends runtime data back to an attached PointeControl Monitor. (Default: 250 msec)
 - **Controller timeout interval** – The time at which the controller will stop execution, if communications with an attached PointeControl Monitor are interrupted and cannot be reestablished. (Default: 5000 msec)
 - **Download source to controller** – The option to download the project's source code to the controller along with the compiled program, so that the project may later be uploaded from the controller back to an attached PointeControl Monitor for debugging.
3. Click **OK** to save your changes and close the window.

For more information on using PointeControl Monitor, see Chapters 6 and 7.

If you are doing a full compile of your project, then proceed to [Checking your project's chart integrity](#).

5.11.3 Checking your project's chart integrity



PointeControl includes a built-in syntax checker that scans your Flow Charts and Ladder Diagrams for missing, incorrect, or undefined tags and block parameters. Some checking occurs as you build your charts, but using the Check Integrity tool ensures that a thorough scan is performed.

The Check Integrity button allows you to locate and correct bugs that prevent the program from compiling correctly. For each identified error in a Flow Chart or Ladder Diagram, the [Messages pane](#) lists the name of the affected chart and the coordinates of the block that contains the error.

NOTE: You can also check the integrity of an individual Flow Chart or Ladder Diagram by clicking the Check Integrity tool in the toolbar of that particular chart.

To check your project's chart integrity:

1. Click the **Check Integrity** tool in the framework editor [toolbar](#), or choose **Check Integrity** from the **Project** menu. If PointeControl detects any errors, they will be displayed in the [Messages pane](#) at the bottom of the Framework Editor window.
2. Double-click a listed error to correct it. PointeControl will open the affected Flow Chart or Ladder Diagram and highlight the block that contains the error.
3. Correct the error.
4. Repeat steps 1 through 3 until all errors are corrected.

5.11.4 Building your project's runtime module



After you have configured your project's [Chart List](#), click the **Build Runtime** toolbar button to build the final runtime module that will be loaded onto the Pointe Controller unit. (You can also choose **Build Runtime** from the **Project** menu.)

As the runtime is built, progress messages are displayed in the [Messages pane](#) at the bottom of the Framework Editor window. If any errors are encountered, then the build process will be aborted. Possible errors include:

- Syntax and chart integrity errors
- I/O configuration errors
- Modbus mapping errors

- Java compilation errors

You must fix all errors before you can try again to build the runtime module.

Once the runtime module is successfully built, proceed to [Activating the PointeControl Monitor](#).

5.11.5 Activating the PointeControl Monitor



After you have built your project's [runtime module](#), click the **Activate Monitor** toolbar button to launch the PointeControl Monitor utility. (You can also choose **Activate Monitor** from the **Window** menu.) This utility manages the downloading of your runtime to the Pointe Controller unit.

For more information on using PointeControl Monitor, see Chapters 6 and 7.

Chapter 6: Downloading to the Controller

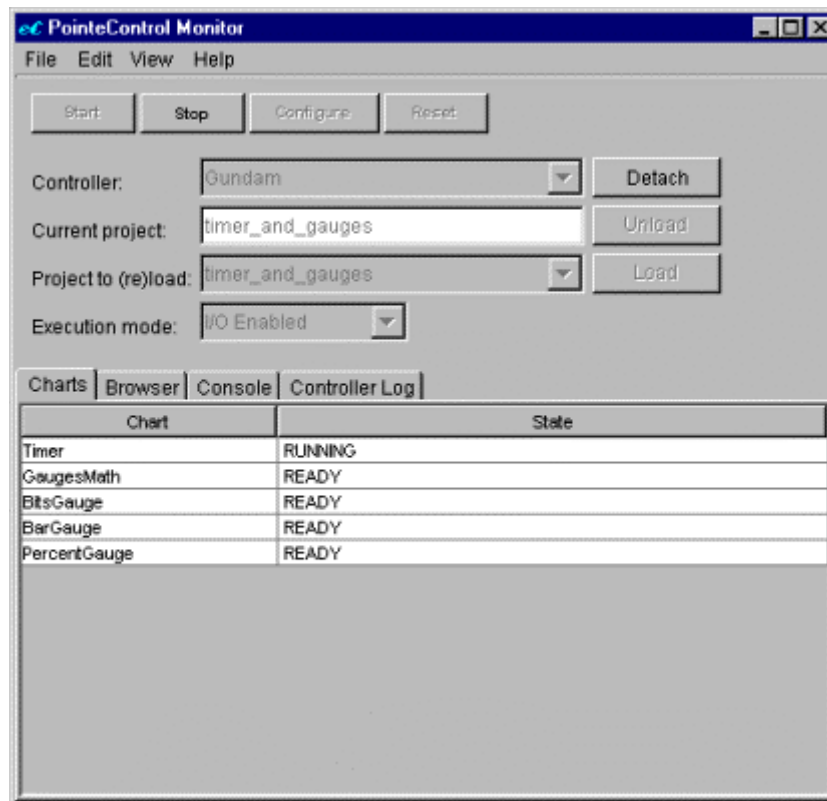
When you have finished developing and compiling your control program, you must use the PointeControl Monitor application to download it to the Pointe Controller unit. PointeControl Monitor (referred to hereafter as PCM) is a Java-based utility for loading, running, monitoring, and debugging projects on Pointe Controller units.

This chapter describes how to select and attach a Pointe Controller unit on the network, then prepare and load a PointeControl project onto the controller.

TIP: The information provided in this chapter is also available via the PointeControl Monitor online help. To access the help, choose **Contents** from the Monitor's **Help** menu.

6.1 Launching the PointeControl Monitor

To launch PCM from within the PointeControl development framework, either click the **Activate Monitor** toolbar button or choose **Activate Monitor** from the **Window** menu. To launch PCM from the Windows Start menu, choose **Start > Programs > PointeControl > Monitor**. When you launch PCM, the PointeControl Monitor window appears:



The PointeControl Monitor window is roughly divided into two halves. The top half of the window is a control panel that is used to download and run finished control programs on the Pointe Controller hardware. The bottom half of the window is a tabbed set of displays that show the status and performance of the program as it runs on the controller. For more information on monitoring program performance, see Chapter 7, "Monitoring and Debugging," starting on page 193.

6.2 Selecting and Attaching a Controller

PCM can interact with any Pointe Controller unit on the local Ethernet network. However, PCM launches in a blank state and must establish an exclusive connection with a specific controller in order for this interaction to occur. Establishing this connection is called “attaching to the controller.”

To select and attach a controller:

1. Click the **Controller** drop-down menu. The menu will list all of the Pointe Controller units that are available on your network. If no controllers are listed, make sure both your target controller and your PC are attached to the same Ethernet network.
2. Select your target controller from the menu. PCM will immediately attempt to attach the controller.

It may take several moments for PCM to successfully attach the controller. This delay occurs as PCM checks to see if a project is already loaded on the controller and, if there is, to secure the corresponding source code for debugging. (For more information, see “[Viewing and debugging charts](#)” on page 205.)

NOTE: If PCM cannot attach the target controller, because of either communication problems or extreme load, then you will be alerted to the failure and an error message will be logged in the [Console tab](#). If the problem persists, you should reset the controller as described on page 230.

Once PCM has attached to the target controller, the control panel will be updated to reflect the current state of the controller:

- If no project is loaded on the controller, then the **Current project** field will remain empty and the **Project to (re)load menu** will become enabled. The **Start**, **Stop**, and **Reset** buttons will be disabled. From here, you can proceed with [loading](#) a project onto the controller.
- If a project is loaded but not running, then project will be displayed in the **Current project** field and the **Start** and **Reset** buttons will become enabled. From here, you can proceed with either [starting](#) the loaded project or [loading](#) a different one.
- If a project is loaded and running, then the project will be displayed in the **Current project** field and the **Stop** button will become enabled. Also, the tabbed displays at the bottom of the window will show the status of the project. From here, you can proceed with either [monitoring](#) the running project or [stopping](#) it.

NOTE: If a password has been set on a Pointe Controller unit, then you must match that password in PCM *before* you can attach the controller. If you do not, you will receive an error message when you attempt to attach. For more information, see “[Assigning a password to the controller](#)” on page 190.

6.2.1 Detaching from a controller

When PCM is attached to a specific Pointe Controller unit, it disables the Controller drop-down menu to prevent you from accidentally disrupting communications with the controller. You must explicitly detach from the currently attached controller before you can select another.

To detach from a controller, click the **Detach** button to the right of the **Controller** drop-down menu. You can then select and attach another controller as described above.

NOTE: If a project is running on an attached controller, it will continue to run after you detach from it.

6.3 Downloading a Project to the Controller

After PCM is attached to a specific Pointe Controller unit, you can select a compiled project from your PointeControl working directory and download it to the controller. Downloading a new project will completely overwrite any existing project on the controller.

To download a project to the controller:

1. Verify that PCM is **attached** to the target controller.
2. If the controller is already running a project, then **stop** it.
3. Click the **Project to (re)load** drop-down menu. The menu will list all of the compiled projects that are available in your PointeControl working directory.
4. Select your desired project from the menu.
5. Click the **Load** button to begin the download process. PCM will prepare the project and download it to the controller. You can watch the process in the **Console tab**.

NOTE: You cannot download the same revision of a project that is already loaded. You must download either a different (newer) revision of the loaded project or different project altogether.

The download process can take several minutes, since it involves converting the compiled project into a program image and writing it to the controller's flash memory. The actual amount of time it takes depends on the processor speed of your PC.

When the load process is finished, the **Start** button will become enabled. You can then proceed to **Starting and stopping a loaded project**.

6.3.1 Unloading a project

Downloading a new project will completely overwrite any existing project on the controller. However, in certain cases (such as sending the hardware back to the vendor for service), you may want to unload the existing project without downloading a new one. To do this, simply attach the controller and click the **Unload** button located to the right of the **Current project** field. This will download a blank (null) project onto the controller, effectively erasing the controller's memory.

6.4 Starting and Stopping a Loaded Project

Once a project is loaded onto the Pointe Controller unit, it's a very easy matter to start the project:

1. Verify that PCM is **attached** to the target controller and your project is **loaded** onto it.
2. From the **Execution mode** drop-down menu, select either **I/O Enabled** or **I/O Disabled**:
 - In **I/O Enabled** mode, the controller will scan all I/O points and execute charts using live data. This option is intended for real control of connected machinery.
 - In **I/O Disabled** mode, the controller will not scan any I/O points and will only execute charts internally using virtual data. This option is useful for debugging logic flow; inputs can be simulated by **forcing tag values**.
3. Click the **Start** button.

When the project is running, you can proceed with **monitoring** and/or **debugging** it.

6.4.1 Stopping a project

To stop a project that is currently running on an attached controller, simply click the **Stop** button. The controller immediately stops scanning charts and I/O, and the project's last state (including tag values and chart conditions) is retained in memory.

WARNING: Stopping a project does *not* stop the controller's internal clock nor any Timers that are keyed to it. Active Timers continue to count elapsed time even while the project is stopped, and all control logic that is based on those Timers will update accordingly when the project is resumed.

To completely stop an active Timer, you must either execute a Timer Stop command (T_STOP) within the project *or* stop *and* reset the project using PCM. For more information on resetting a project, see below.

6.4.2 Restarting a stopped project

To resume a stopped project from its last state, simply click the **Start** button again.

To restart a stopped project from its initial state – that is, the initial tag values and chart conditions that you defined when you created the project in the PointeControl Framework – click the **Reset** button and then the **Start** button.

6.4.3 Enabling and disabling I/O

You can change the controller execution mode (i.e., whether real I/O scanning is enabled or disabled) any time the project is stopped – for example, if you began your debugging with I/O disabled and now want to enable it. To change the execution mode:

1. Verify that PCM is attached to the target controller and your project is loaded and running.
2. Click the **Stop** button.
3. Click the **Execution mode** drop-down menu and select either **I/O Enabled** or **I/O Disabled**, as desired.
4. If you wish to restart the project from its initial state, click the **Reset** button.
5. Click the **Start** button.

6.5 Assigning a Password to the Controller

You can secure one or more Pointe Controller units against unauthorized access by assigning passwords to them. You can assign the same password to a group of controllers by first creating the password in PCM and then assigning it to each controller in the group.

Once a password is assigned to a controller, you must have that password set in PCM *before* you can reattach the controller. (If you do not, you will receive an error message.) The password is retained in memory, allowing you to freely attach all matching controllers.

To create/set a password in PCM:

1. From the **Edit** menu, choose **Password**. The **Controller group password** window will appear.
2. Enter your password, then tab to the next field and enter it again to confirm.
3. Click **OK**. The password is retained until a new password is set or until the current PCM session is closed.

Once the password is set in PCM, it can be used both to add new, unsecured controllers and to access controllers that are already secured with that password.

To assign the current password to an unsecured controller:

1. Verify that PCM is **attached** to the target controller.
2. From the **Edit** menu, choose **Join group**. The password will be assigned to the controller and the change will be confirmed in the **Controller Log**.

NOTE: The password is saved permanently in the controller's flash memory, until the controller is removed from the group or undergoes a hard reset (see below).

3. To assign the same password to other controllers, **detach** from the current controller and repeat the process for each additional controller.

To clear the password from a secured controller:

1. Verify that PCM is **attached** to the target controller. (You must have the appropriate password set in PCM before you can attach the controller.)
2. From the **Edit** menu, choose **Leave group**. The password will be erased from the controller and the change will be confirmed in the **Controller Log**.
3. To clear the same password from other controllers, **detach** from the current controller and repeat the process for each additional controller.

6.5.1 Overriding a password

If you cannot remember a controller's password, you can bypass it via hardware override. To override the password on a controller:

1. Power off the affected controller.
2. Change the controller's Modbus address to "98," using the rotary switches located on the controller's motherboard.
3. Power on the controller.
4. Launch PCM, attach the controller, and clear the password.
5. Power off the controller.
6. Change the Modbus address back to its original setting.
7. Power on the controller.

For more information on accessing the Pointe Controller rotary switches, see "Hardware Reset" on page 230.

6.6 Saving a Project from the Controller

If you wish to edit a project that was not developed on your own PC, you can retrieve the project's source code from the controller and save it in your PointeControl working directory. From there, it can be opened normally in the PointeControl development framework.

To save a project from the controller:

1. Verify that PCM is **attached** to the target controller.
2. From the **File** menu, choose **Save Current Project As**. The save file dialog will appear.
3. Enter a name for the project.
4. Click **OK**. The project is saved.

NOTE: In order to retrieve a project's source code from the controller, the project must have had the **Download source to controller** preference enabled when the project was originally loaded. The preference is enabled by default, but some developers may disable it to restrict access to the project.

Chapter 7: Monitoring and Debugging

One of the powerful features of PointeControl is the ability to monitor and debug program operations while they are running on the Pointe Controller unit.

This chapter is divided into two sections:

- **Monitoring a running project** – This section describes how to use PCM's built-in tools to monitor the status and performance of the project while it is running on the Pointe Controller unit.
- **Viewing and debugging charts** – This section describes how to use PCM's built-in tools to debug the project's Flow Charts and Ladder Diagrams, by manipulating individual tag values and block execution.

TIP: The information provided in this chapter is also available via the PointeControl Monitor online help. To access the help, choose **Contents** from the Monitor's **Help** menu.

7.1 Monitoring a Running Project

After you have **loaded** and **started** a project on your Pointe Controller unit, you can monitor the project's behavior using the tools included in PCM. These tools are accessible through the four tabbed panes along the bottom of the PCM window:

- The **Charts tab** lists all of the Flow Charts and Ladder Diagrams that are running in the current project. You can select any listed chart to open it for viewing and/or debugging.
- The **Browser tab** provides a searchable list of all the Logic Memory tags, strings, and timers in the current project. You can select individual tags to see their real-time values or to force new values.
- The **Console tab** displays all status and error messages generated by PCM itself as it communicates with attached controllers.
- The **Controller Log tab** shows the activity log of the currently attached controller. Logged activities include project loads and unloads, project starts and stops, and password group changes.

You can also check the general system performance (scanning speed, processor usage, I/O errors) of the Pointe Controller itself by using the **Performance Metrics** window.

7.1.1 The Charts Tab

Chart	State
Timer	RUNNING
GaugesMath	READY
BitsGauge	READY
BarGauge	READY
PercentGauge	READY

The **Charts** tab displays all of the Flow Charts and Ladder Diagrams that are running in the current project.

NOTE: In this chapter, the term “charts” refers to both Flow Charts and Ladder Diagrams collectively.

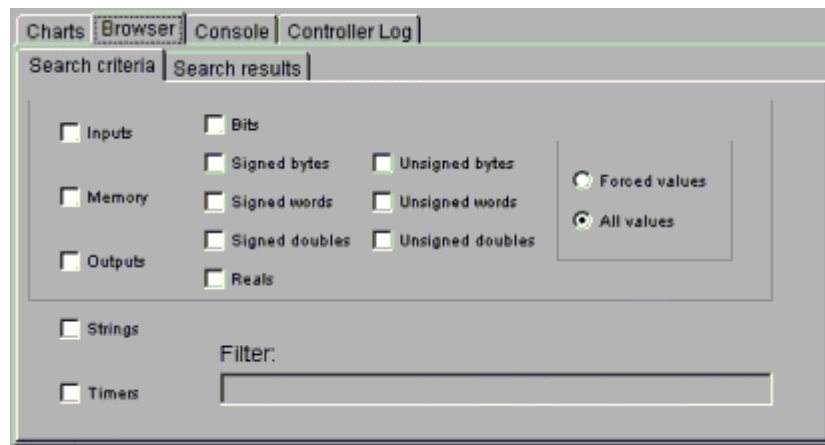
The tab displays up to four columns:

- **Chart** – The name of the chart.

- **State** – The current state (READY, RUNNING, etc) of the chart.
- **Block** (optional) – If **logic flow** is enabled in the debugger, then this column shows which block in the chart is currently being executed.
- **Execution Time** (optional) – If diagnostic timers are enabled in the debugger, then this column shows the total time to execute the chart.

To open any chart for viewing and/or debugging, simply double-click on it. The chart will be opened into a new debugger window. For more information, proceed to [“Viewing and debugging charts”](#) on page 205.

7.1.2 The Browser Tab

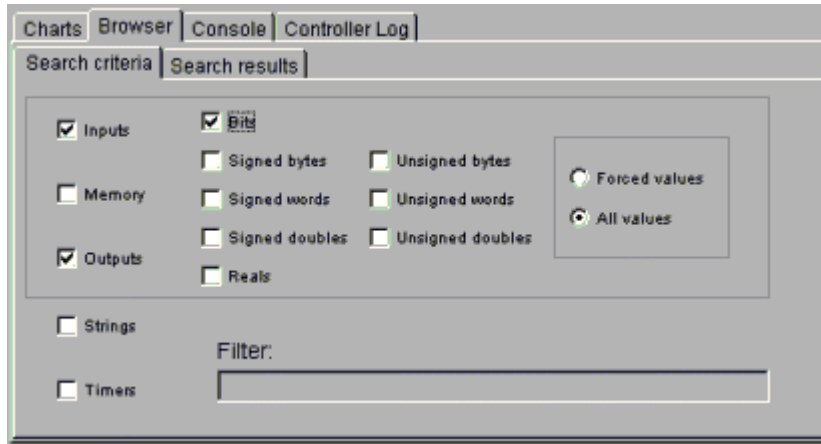


The **Browser** tab provides a searchable list of all the Logic Memory tags, strings, and timers in the current project. You can select individual tags to see their real-time values or to force new values.

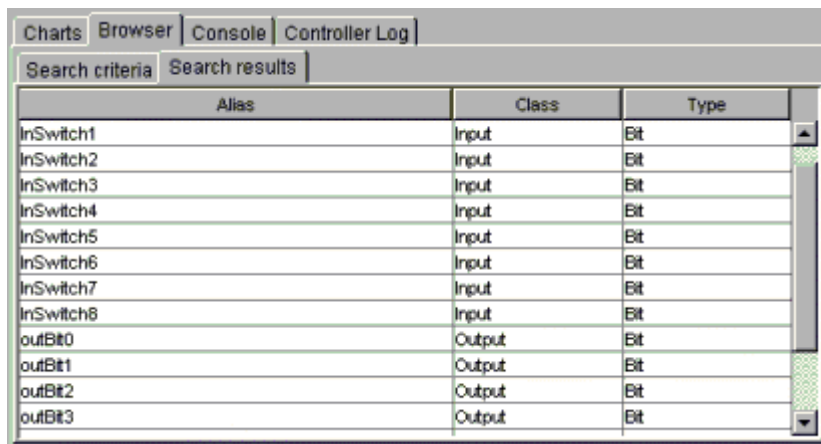
To create a new search:

1. Click the **Search criteria** tab.

2. Select the classes and types of tags for which you want to search by clicking the corresponding checkboxes. For example, to search for all Input Bit and Output Bit tags, click the **Inputs**, **Outputs**, and **Bits** checkboxes:



3. Click the **Search results** tab. The tags that match the selected types will be listed.

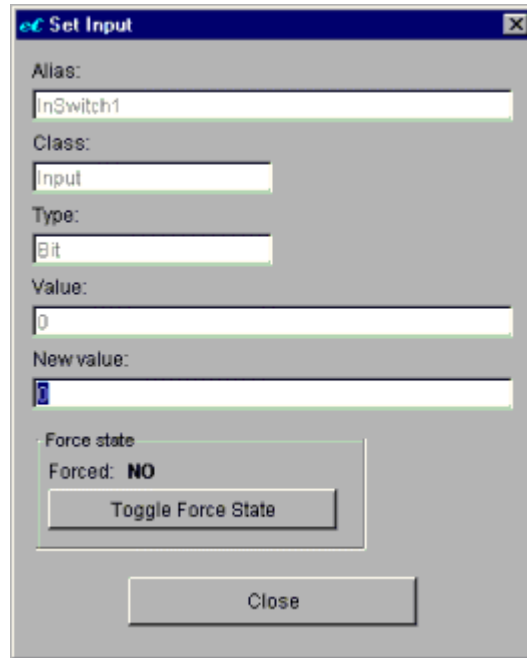


There are two ways to narrow your search even further:

- Switch from **All values** to **Forced values** to search only for tags which you have manually forced to new values.
- Enter a **Filter** string to search only for tag names that start with that string. For example, a Filter string of "InSwitch will return InSwitch1, InSwitch2, InSwitch3, and so on.

Showing tag details

Once you have your search results, you can double-click on any listed tag (or right-click and choose **Show Details**) to open a new window that shows detailed information about the tag:



From this window you can also force the tag to a new value – but for more information on that, see [“Forcing new tag values”](#) on page 212.

Adding tags to a watch window

As you monitor your running project, you can build a **Watch Window** that shows only your favorite tags. To add a tag from the Browser search results to the Watch Window, right-click on the desired tag and choose **Add to watch window**.

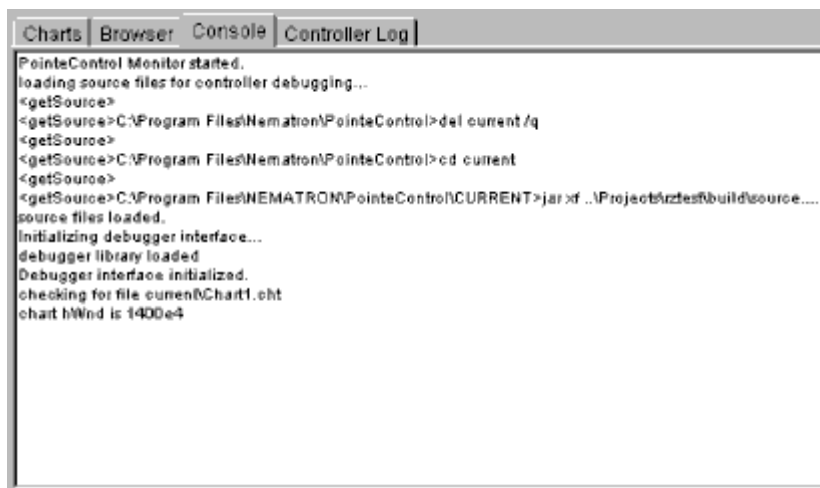
Alias	Class	Type	Value
dataAvailable	Input	Bit	0
InSwitch1	Input	Bit	0
outBit0	Output	Bit	1
outBit1	Output	Bit	1
outBit2	Output	Bit	0
outBit3	Output	Bit	1
outBit4	Output	Bit	0
outBit5	Output	Bit	0
outBit6	Output	Bit	0
outBit7	Output	Bit	0
counter	Timer		4646
displayPercent	String		4

Just as in the Search results tab, you can double-click on any listed tag to open a new window that shows detailed information about the tag.

After you have built your Watch Window, you can save its configuration as a .WWD file in your project's working directory. You can do this either by choosing **File > Save** or by closing the window and clicking **Yes** when prompted. Then the next time you run PCM and monitor the project, you can quickly restore the Watch Window to its saved configuration by choosing **File > Open** and selecting the desired .WWD file.

NOTE: You can open the Watch Window directly from the main PCM window by choosing **View > Watch Window**.

7.1.3 The Console Tab



The **Console** tab displays all status and error messages generated by PCM as it communicates with attached controllers. Types of messages displayed here include:

- Program activity messages for the PCM application itself;
- Communications status and error messages, as PCM communicates with the Pointe Controller unit via the Ethernet network;
- Progress messages generated as a finished PointeControl project is prepared for loading onto the controller; and
- Debugging status messages, as PCM's built-in debugger interfaces with the project running on the controller.

Saving or clearing messages

Messages can be saved from the tab by selecting (highlighting) the desired messages, right-clicking on the selection, and choosing either **Copy** or **Save As** from the pop-up menu. If you choose **Copy**, then you can paste the selection into

another application such as MS Word or Notepad. If you choose **Save As**, then you will be prompted to save the selection as a text file.

If no specific messages are selected, then the entire backlog will be copied/saved.

You can also clear the log by right-clicking and choosing **Clear**.

Communications errors

The table below lists the possible communications error messages:

Code	Name	Description
1	TNT_INV_ARGUMENT	An invalid argument was passed to the method
2	TNT_INV_CLASS	An invalid class name was passed to the method
3	TNT_NOT_OPEN	A TNTManager object has not been created
4	TNT_TOO_MANY_MGR	Current number of TNTManager's at maximum
5	TNT_BASE_SUPT	Unable to initiate Paragon services
6	TNT_BIN_FILE	Unable to locate Client Objects BIN file
7	TNT_INV_HANDLE	Invalid handle used for receiver
8	TNT_INV_METHOD	Method is not supported by object
9	TNT_TRACE_FAIL	Unable to open trace file
10	TNT_NO_MEMORY	Unable to allocate required memory
11	TNT_IMPROPER_METHOD	Method is inappropriate given current object state
12	TNT_TRC_NO_MEM	Unable to trace due to inability to allocate memory
13	TNT_CRA_OPEN_ERR	CRA error - unable to open stream
14	TNT_CRA_ADD_ERR	CRA error - unable to add request
15	TNT_CRA_INFO_ERR	CRA error - unable to perform information query
16	TNT_READ_ERR	CRA error - unable to perform read

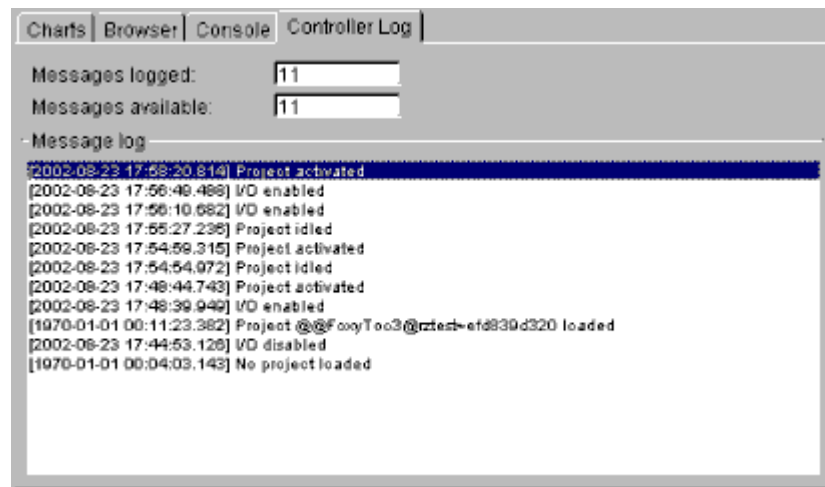
Code	Name	Description
17	TNT_WRITE_ERR	CRA error - unable to perform write
18	TNT_READWRITE_ERR	CRA error - unable to perform read/write (query)
19	TNT_CRA_NAMES_ERR	CRA error - unable to retrieve names
20	TNT_INV_DEFINITION	Invalid structure definition
21	TNT_INV_REFERENCE	Invalid structure reference
22	TNT_NOT_AVAILABLE	Data is not available at this time
23	TNT_CRA_GETNAMES_ERR	CRA error - unable to access directory
24	TNT_ARCHIVE_OPEN_FAIL	Unable to open archive file
25	TNT_ARCHIVE_WRITE_FAIL	Unable to write into archive file
26	TNT_ARCHIVE_READ_FAIL	Unable to read from archive file
27	TNT_NO_CRA	Client Objects initialized without CRA support

To fix a communications error, try the following:

- Wait for the transient condition to clear and try again to attach the controller.
- Verify that both the Pointe Controller unit and your PC are properly connected to the Ethernet network. Also verify that their IP addresses and other network settings are properly configured.
- Cycle power to the Pointe Controller unit.

If none of these fix the error, then please contact Nematron Customer Support.

7.1.4 The Controller Log Tab



The **Controller Log** tab shows the activity log of the currently attached controller. Logged activities include project loads and unloads, project starts and stops, password group changes, and so on. The 25 most recent messages are retained in the controller’s flash memory.

For more information on interpreting controller log messages, see Chapter 9, “Troubleshooting,” starting on page 227.

Saving or clearing messages

Messages can be saved from the tab by selecting (highlighting) the desired messages, right-clicking on the selection, and choosing either **Copy** or **Save As** from the pop-up menu. If you choose **Copy**, then you can paste the selection into another application such as MS Word or Notepad. If you choose **Save As**, then you will be prompted to save the selection as a text file.

If no specific messages are selected, then the entire backlog will be copied/saved.

You can also clear the log by right-clicking and choosing **Clear**.

7.2 Checking System Performance

You can check the system performance of an **attached** Pointe Controller unit by opening the Performance Metrics window:

- From the **View** menu, choose **System Performance**. The Performance Metrics window will appear.

The Performance Metrics window offers three tabs: Scanning, Loading, and Errors.

7.2.1 Scanning

	Average	Maximum	Minimum	Overruns
Chart/Ladder	6005	8462	5409	14
Module 1	1043	1213	996	2
Module 2	1025	1143	996	2
Module 3	159	1307	63	1
Module 4	160	1419	63	1
Module 5	0	0	0	0
Module 6	0	0	0	0
Module 7	0	0	0	0
Module 8	0	0	0	0
Panel	2530	2786	2432	0
RTU 1	0	0	0	0
RTU 2	0	0	0	0
RTU 3	0	0	0	0
RTU 4	0	0	0	0
Chart & I/O Total Scans	278598	1393171		
Chart & I/O Scan Interval	10	2		

all times in microseconds

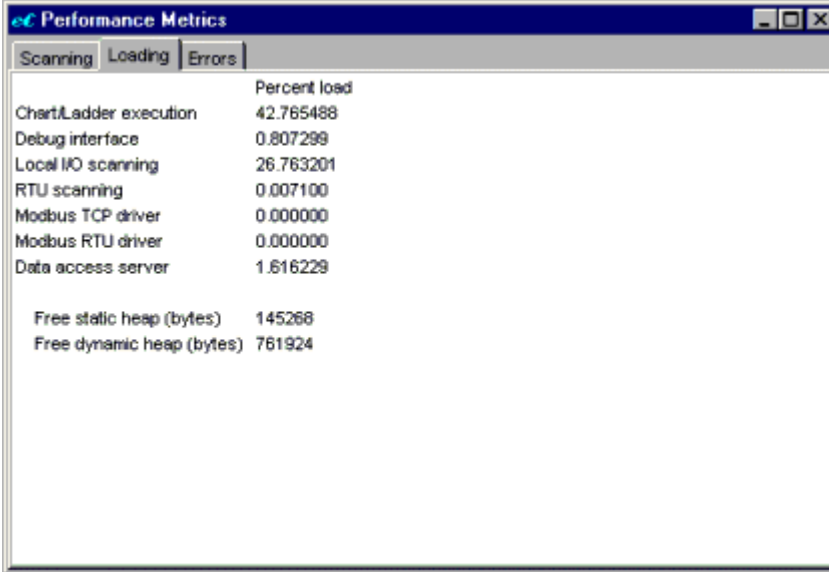
The Scanning tab shows how much real time (in microseconds) it is taking the controller to process the Flow Charts and Ladder Diagrams that make up the currently running project, as well as to scan the I/O modules, operator panel, and OptiLogic RTUs that are connected to the controller.

For charts/ladders, the scan time is the time elapsed from when the input tags are read in, through the entire logic solve, to when the output tags are written out. For modules, panels and RTUs, the scan time is the time elapsed from when the controller's processor becomes available to scan the component to when that component's scan is complete.

Overruns occur when a component does not complete its current scan before its next scheduled scan. For example, if a module is configured with a Scan Interval of 10, then it is scheduled to be scanned at 0, 10, 20, 30, and so on. The scan starting at 0 must be completed before 10. If it is not, it is registered as an overrun.

NOTE: All charts and I/O scan concurrently, not sequentially.

7.2.2 Loading



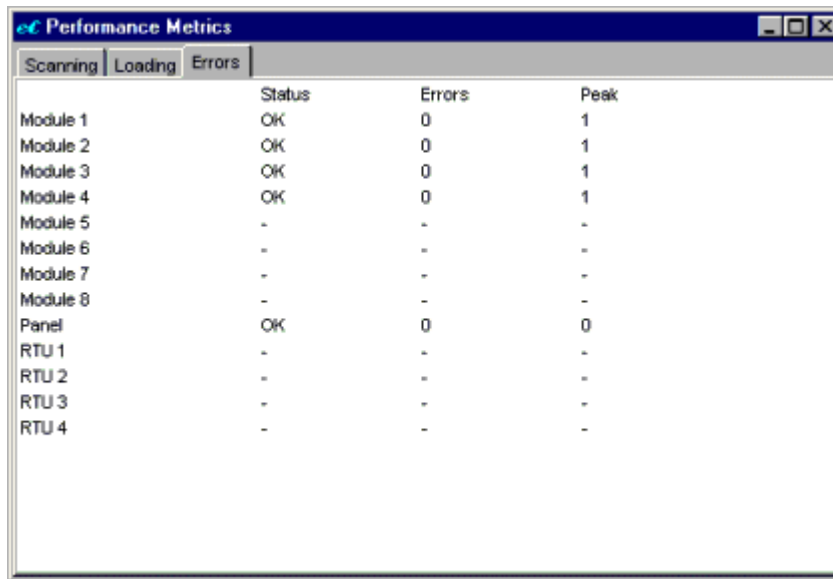
	Percent load
Chart/Ladder execution	42.765488
Debug interface	0.807299
Local I/O scanning	26.763201
RTU scanning	0.007100
Modbus TCP driver	0.000000
Modbus RTU driver	0.000000
Data access server	1.616229
Free static heap (bytes)	145268
Free dynamic heap (bytes)	761924

The Loading tab shows how much load each PointeControl task is putting on the controller's processor. Each task's load is expressed as a percentage of the processor's total available resources:

- **Chart/Ladder execution** is the load to process the Flow Charts and Ladder Diagrams that make up the currently running project.
- **Debug interface** is the load to interface with the Monitor's [debugging tools](#).
- **Local I/O scanning** is the load to scan and update the I/O modules and operator panel specifically in the attached controller.
- **RTU scanning** is the load to scan the I/O points on any additional Remote Terminal Units (RTUs) that are attached to the controller using the OptiLogic UDP/IP protocol.
- **Modbus TCP driver** is the load to maintain the controller's built-in Modbus TCP driver, if enabled, and to respond to incoming Modbus requests.
- **Modbus RTU driver** is the load to run the controller's built-in Modbus RTU driver, if enabled, and to respond to incoming Modbus requests.
- **Data access server** is the load to maintain communications between the controller and the Monitor. This includes support for tag browsing, controller log, performance metrics, and network discovery.

Free static heap indicates how much non-reusable memory is still available. **Free dynamic heap** indicates how much reusable memory is still available. The dynamic heap will decrement to zero as memory is used and reset as memory is recovered.

7.2.3 Errors



	Status	Errors	Peak
Module 1	OK	0	1
Module 2	OK	0	1
Module 3	OK	0	1
Module 4	OK	0	1
Module 5	-	-	-
Module 6	-	-	-
Module 7	-	-	-
Module 8	-	-	-
Panel	OK	0	0
RTU 1	-	-	-
RTU 2	-	-	-
RTU 3	-	-	-
RTU 4	-	-	-

The Errors tab shows the current status of the I/O modules, operator panel, and OptiLogic RTUs connected to the controller, as well as the total number of communications errors encountered by each device since the current project was started.

Status is the current operating condition of the given module. **Errors** is the current "streak of consecutive errors. **Peak** is the longest streak encountered since the project was started.

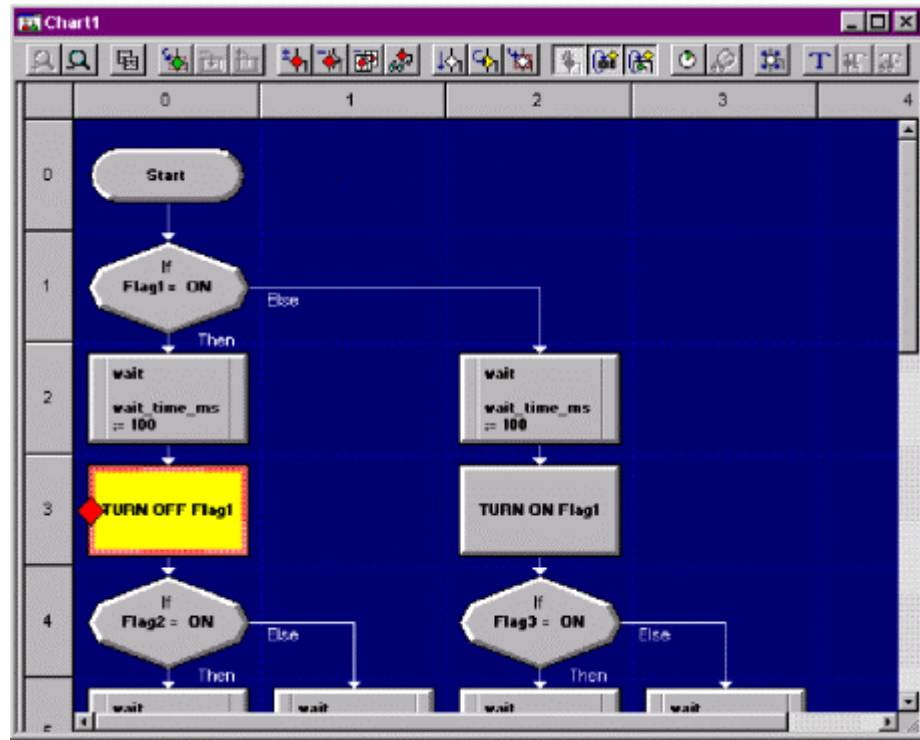
Errors on a module or panel are indicative of serial communication errors on the controller's motherboard. A single error or two may be encountered during initialization. Continuous errors may be indicative of a hardware defect or a poorly seated connection.

Errors on an RTU are more likely an indication of problems in the Ethernet network – either a bad network connection or excessive network load.

For more information, see Chapter 9, "Troubleshooting," starting on page 227.

7.3 Viewing and Debugging Charts

You can open any chart for viewing and debugging by double-clicking on the chart in the **Charts** tab. The chart will be opened into a new Debugger window:

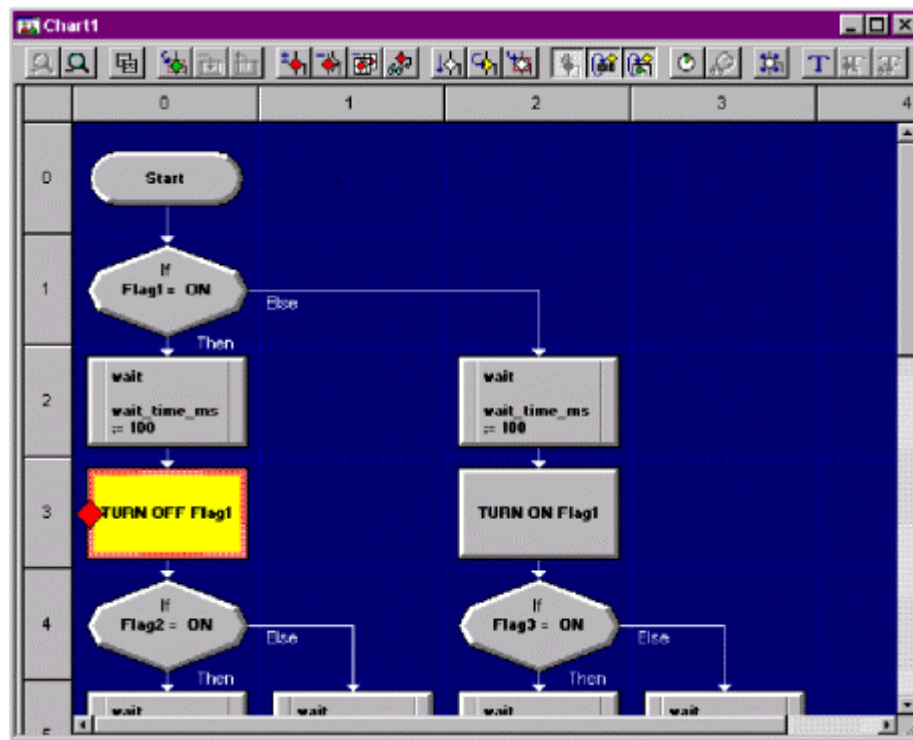


The Debugger window displays a real-time view of the selected chart’s activity and provides tools to interactively control the program flow. The Debugger window highlights the currently active block. As the program flows, different blocks appear highlighted as they become active.

For more information on the different parts of the Debugger window, proceed to [“The Debugger window”](#) on page 206.

NOTE: When PCM first attaches to a Pointe Controller unit, it attempts to retrieve the source code of the currently loaded project. The source code is required for debugging, and it can also be **saved** locally for further editing. By default, the source code is always loaded onto the controller along with the compiled project. However, this can be prevented by disabling the **Download source to controller** preference in the PointeControl development framework. If the source code cannot be retrieved, then the project cannot be debugged.

7.3.1 The Debugger Window



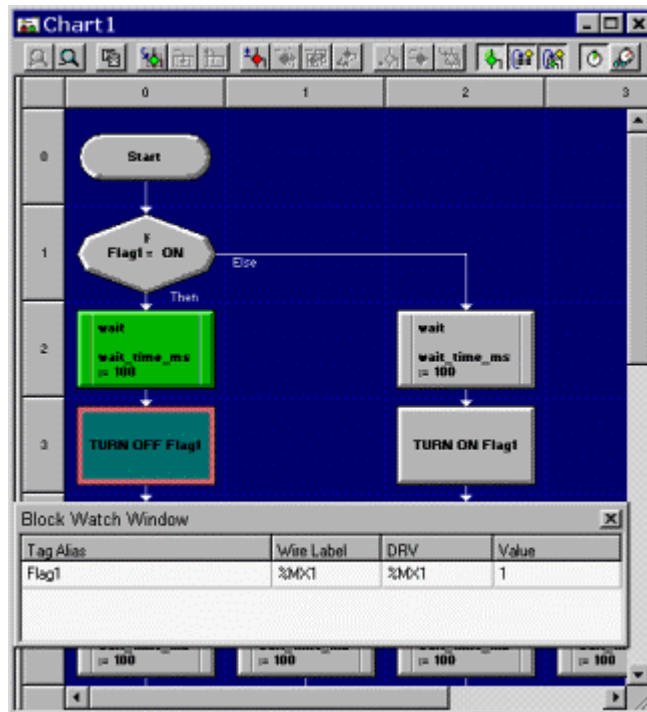
The Debugger window displays a real-time view of the selected chart's activity and provides tools to interactively control the program flow. The currently active block is always highlighted; as the program flows, different blocks are highlighted as they become active.

A toolbar containing icons for each of the tools available to the debugger appears below the title bar. Most of the tools are usable in both Flow Charts and Ladder Diagrams, but **some tools are for Flow Charts only**. Each tool is described later in this document.


Below the toolbar is a workspace area in which the chart is displayed. Scroll bars will appear as needed to view the diagram, based on **magnification level** and size of the diagram.

The cursor is always in Select mode within the Debugger window. Any chart block may be selected by a single-click. Object selection enables some of the tools in the toolbar and disables others.

To view the activity of a particular block in the chart displayed in the Debugger window, double-click on that block. A **Block Watch Window** for that block will be displayed. The window lists all of the Logic Memory tags, strings, and timers that are read or updated by the selected block. The value of each tag is dynamically updated as the value changes within the program flow.




7.3.2 Zooming In and Out on a Chart

 A chart that is opened in the Debugger window can be viewed at several different magnification levels using the **Zoom In** and **Zoom Out** tools. Zooming is useful when you are trying to view a especially large chart and you need to see how the logic flows without scrolling around.

Zoom In makes the chart appear larger, while Zoom Out makes the chart appear smaller, allowing you to see more of it within the workspace window. By default, a chart is initially displayed at the maximum magnification level.

7.3.3 Viewing Subcharts within a Chart


 When a chart contains a call to another subchart, the subchart can be viewed by selecting the calling block and clicking the **Open Subchart** tool (or **Open Subdiagram** in ladder). The subchart will be opened, replacing the previous chart in the active Debugger window. All of the regular debugging tools are available while viewing the subchart.

 Once a subchart is opened, it can be closed again by clicking the **Close Subchart** tool (or **Close Subdiagram** in ladder). The previous chart will be

restored in the Debugger window, with magnification and scroll position intact.

NOTE: The Open Subchart tool is enabled only when a calling block (Subchart block in Flow Charts, CALL function block in Ladder Diagrams) is selected. The Close Subchart tool is enabled only when the subchart is open in the Debugger window.

7.3.4 Enabling Logic Flow in a Chart


 Clicking the **Enable Logic Flow** tool suspends the normal execution cycle on the Pointe Controller unit and enters the Logic Flow debugging mode. When Logic Flow is enabled (the button remains depressed), the controller checks for a **breakpoint** at each chart block transition, pauses if it finds a breakpoint, and waits for the command to **continue**.

Logic Flow *must* be enabled in order to insert breakpoints. Also, when breakpoints are set, the Enable Logic Flow toolbar button is locked to ensure that all breakpoints are honored.


Enabling logic flow also enables the Block column in the **Charts tab**.

NOTE: Checking for and processing breakpoints causes a significant impact on the performance of the controller. As such, Logic Flow is automatically disabled when all breakpoints are removed and all debugger windows are closed.

Scan I/O During Single Step Mode


 When **Scan I/O During Single Step Mode** is enabled (the tool button remains depressed), the controller continues to scan all of its I/O points even while one chart is stopped on a breakpoint. Disabling this mode (the tool button is not depressed) prevents I/O scanning. This mode is enabled by default, but it is useful to disable it if you do not want the controller to run unsupervised while you are working with a specific breakpoint.

Run Charts During Single Step Mode

 When **Run Charts During Single Step Mode** is enabled (the tool button remains depressed), the controller continues to execute all other charts even while one chart is stopped on a breakpoint. Disabling this mode (the tool button is not depressed) prevents the execution of other charts. This mode is enabled by default, but it is useful to disable it if you do not want the controller to run unsupervised while you are working with a specific breakpoint.

NOTE: Since these tools control the actual state of the controller, they are automatically synchronized across all debugger windows.

7.3.5 Enabling Debug Trace in a Chart


 Selecting the **Enable Debug Trace** tool activates an internal tracing of how the chart's logic flow is executed. The trace is dynamically updated as each

block is executed, and the information can be displayed either as a visible path in the debugger window (Show Debug Trace) or as tabular data (View Debug Trace).

Up to 1000 steps (records) can be saved in a circular memory buffer; if more than 1000 steps are recorded, the trace overwrites from the beginning of the buffer. The buffer is reset for each new scan of the chart.

NOTE: Enabling Debug Trace adversely affects the execution speed of the Pointe Controller unit. As such, Debug Trace is automatically disabled when the debugger window is closed.


Show Debug Trace

 Once debug tracing is enabled, you can select the **Show Debug Trace** tool to display the tracing in the debugger window. In Flow Charts, the trace is displayed as a yellow frame around the currently active block. In Ladder Diagrams, the trace is displayed as a green highlight along the currently active rung.

Show Debug Trace is automatically enabled when the Enable Debug Trace tool is selected.

NOTE: In Ladder Diagrams, the trace display and trace record collections are interlocked so that a whole pass through the diagram is shown. That is, the display of the trace will be done only after the last ladder object in the last rung of the diagram executes, but before the trace buffer is reinitialized at the top of the diagram. This differs from the tracing done in the Flow Charts, in which partial passes through a chart are shown.

View Debug Trace

 In addition to the visual debug trace, a tabular readout of the trace records can be displayed by selecting the **View Debug Trace** tool. Trace records are listed from newest to oldest, and each record shows the name of the chart (or called subchart) and the object's coordinates (rung/block). A sample table is shown below:



TID	Chart Name	Block [x,y]
2	Ladder3	[2, 3]
2	Ladder3	[2, 2]
2	Ladder3	[2, 1]
1	Ladder3	[1, 3]
1	Ladder3	[1, 2]
1	Ladder3	[1, 1]

The trace records can also be printed or saved to a file for subsequent analysis.

7.3.6 Inserting Breakpoints in a Chart



Breakpoints may be set and cleared on selected blocks using the **Insert/Remove Breakpoint** tool. Selection of a block enables the tool. Clicking the tool toggles the breakpoint status of the block.

NOTE: Breakpoints can be inserted only when **Logic Flow is enabled**.

When a breakpoint is set, the block is shown with a red background. All logic flow in the current project pauses the next time the block is about to execute, before any processing occurs for the block. If the Debugger window containing the breakpoint is closed or hidden when the breakpoint is reached, it is opened and/or brought to the front.

When logic flow pauses on the breakpoint, the block is shown with a yellow background. When the breakpoint is cleared, the block is shown with a green background.

Use the **Go**, **Single Step**, or **Run to Cursor** tools to continue execution of logic flow.

Setting a breakpoint on any block will also enable the following tools...

Remove All Breakpoints



All breakpoints set within the active (foreground) Debugger window can be cleared by clicking the **Remove All Breakpoints** tool. All blocks containing breakpoints will be restored to their normal (green) state. However, removing the breakpoints does not re-enable execution flow – use the Go, Single Step, or Run to Cursor tools to resume execution (see above).

Remove All Breakpoints in All Threads



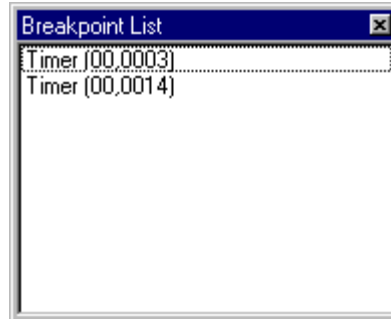
All breakpoints set in *all charts* in the project can be cleared by clicking the **Remove All Breakpoints in All Threads** tool. All blocks containing breakpoints will be restored to their normal (green) state. However, removing the breakpoints does not re-enable execution flow – use the Go, Single Step, or Run to Cursor tools to resume execution (see above).

Display Breakpoint List




Breakpoints established in any chart can be viewed using the **Display Breakpoint List** tool. This tool is enabled whenever any breakpoints are set

within the executing OpenControl project. Selecting the tool activates the Breakpoint List dialog. This dialog will float on top of all OpenControl runtime windows, so selecting another debugger window or the OC Monitor will not hide the breakpoint list. An example breakpoint list is shown below:




Each breakpoint in the list shows the name of the chart (followed by the subchart name for breakpoints in **subcharts**) and the coordinates of the affected block. Double-clicking on a listed breakpoint will open the chart in which the breakpoint is set.

7.3.7 Continuing Execution after a Breakpoint

 Once a **breakpoint** is reached, all logic flow in the chart containing the breakpoint stops. Execution of the chart can be resumed by clicking the **Go** tool and it continues normally until the next breakpoint is reached.

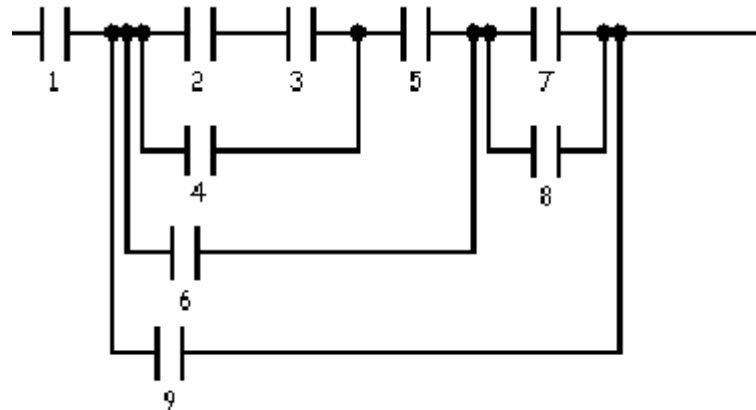
NOTE: By default, all other charts and I/O in a project continue to scan while one chart is stopped on a breakpoint. For more information, see [“Enabling logic flow in a chart”](#) on page 208.

Single Step

 Another option is to execute a single block, or “step, by selecting the **Single Step** tool. Clicking this tool places a temporary breakpoint on the next block to execute and then runs *only* to that block. The temporary breakpoint is cleared when reached and it does not appear in the breakpoint list.

You can click the Single Step tool repeatedly to continue “stepping through chart, one block at a time.

NOTE: In Ladder Diagrams, branches are executed according to a special ordering algorithm. For example:



The numbers indicate the order in which the contacts are executed. As is shown, the ordering is from innermost branch to outermost branch. The main branch is followed until a merge point, at which the preceding split point is found and its branch followed.

Run to Cursor



A third option is to execute all the way up to a specific block in the chart, by selecting the desired block and clicking the **Run to Cursor** tool. A temporary breakpoint is set on the selected block and the chart is executed only up to it. The temporary breakpoint is cleared when reached and it does not appear in the breakpoint list.

When using this tool, be aware that the breakpoint set on the selected block may never be reached if, depending on the flow of the chart, the block is never executed. In such a case, the temporary breakpoint will remain set until manually removed using the [Remove Breakpoint](#) tool.

7.3.8 Forcing New Tag Values

When you double-click on any tag listed in the Browser tab or Watch Window, a pop-up window is displayed that allows you to force a new value for the tag. The new value is treated as real data by the Pointe Controller unit.

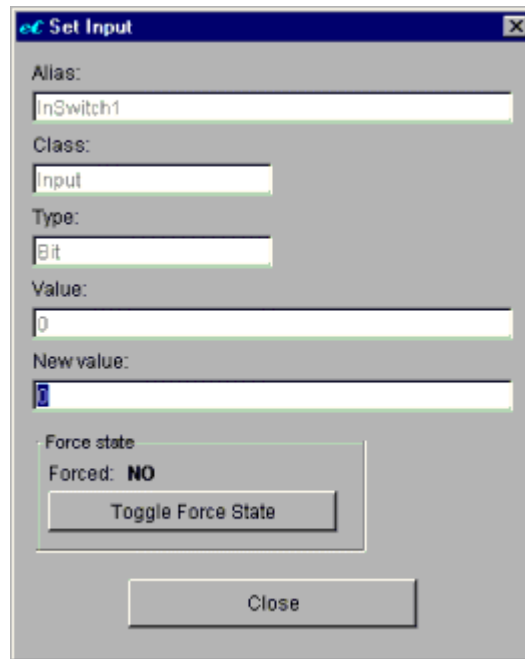
This is useful for testing hypothetical conditions that may not be reached in the normal execution of your project. It can also be used to simulate inputs and outputs when you are running your project with [I/O disabled](#).

WARNING: Please exercise extreme caution when using this function with I/O enabled. Forcing a value while I/O is enabled can cause connected equipment to exceed normal operating parameters.

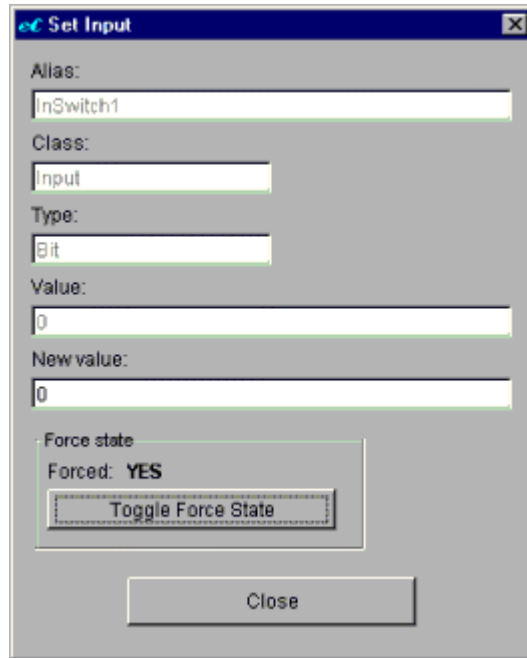
Input, Output, and Memory tags

To force a new value for an Input, Output, or Memory tag:

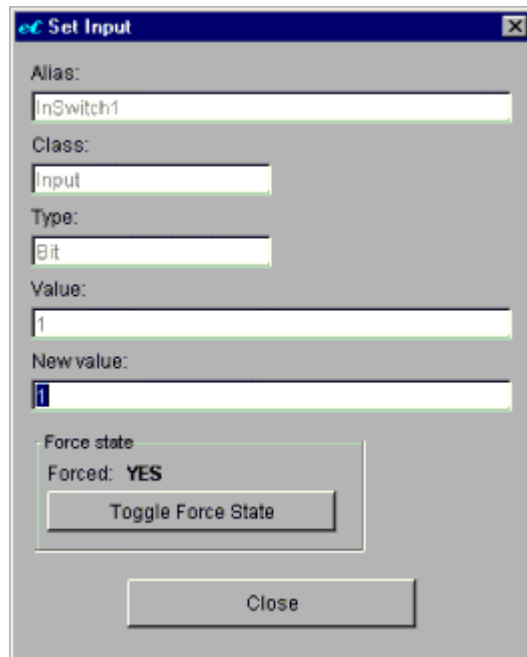
1. Open the tag by double-clicking on it in the **Browser tab**, in the main **Watch Window**, or in an individual **Block Watch Window**. The tag's detail window will be displayed.



2. Click the **Toggle Force State** button. This will lock the tag against updates that may be made by the normal execution of your project.



3. Enter the **New value** and press return. The tag will be immediately changed from its previous value to the new value.








To remove the force, simply click the **Toggle Force State** button again. The value will then be overwritten by the normal execution of the project.

NOTE: You can enter a new value without toggling the tag’s force state, but then the tag can be overwritten.

Strings and Timers

Strings and timers differ from tags only in that you do not need to click the Toggle Force State button before entering a new value. (In fact, there is no such button in the string and timer windows.) Whatever value you enter will be accepted immediately by the project. However, the value cannot be locked and will be overwritten by the normal execution of your project.

7.3.9 Additional Tools for Flow Charts Only

Button	Tool	Description
	Toggle Labels	Toggles the display of block labels, as determined by each block’s Caption property.
	Select Active Block	Directly selects the block that is currently being executed.
	Enable Diagnostic Timers	Enables diagnostic timers that measure the total time to execute charts, as well as time spent executing loops. Selecting this tool also enables the Execution Time column in the Charts tab .
	View Diagnostic Timers	Opens a window showing all of the diagnostic timers for the current chart. Diagnostic timers must already be enabled as described above. Since only times of 1 second or more are displayed, this window is usually empty.
	Size to Content	Adjusts the sizes of the blocks in the current chart to accommodate the content of the blocks. Resizing is not persistent; blocks will revert to their original sizes when the debugger window is closed.

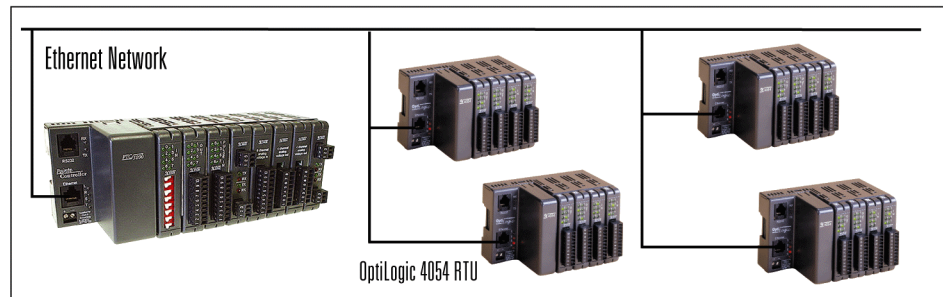
Chapter 8: Networked Operations

In addition to operating as a stand-alone machine controller, the Pointe Controller unit can also be integrated into a networked control system. It can serve as supervisory node, administering up to four remote I/O terminals, or it can be configured as a Modbus slave device in a larger Modbus network.

TIP: The information provided in this chapter is also available via the PointeControl Framework online help. To access the help, choose **Contents** from the Framework's **Help** menu.

8.1 Networking via OptiLogic Remote I/O

As a Master Controller for distributed control solutions, the Pointe Controller is capable of interfacing with up to four OptiLogic I/O terminals via Ethernet. This reduces wiring cost of the I/O devices back to the controller, while providing high speed I/O control from the controller.



You can configure up to four additional OptiLogic Remote Terminal Units (RTUs) to work with your Pointe Controller unit. These units are slaved to your controller using the OptiLogic UDP/IP communication protocol.

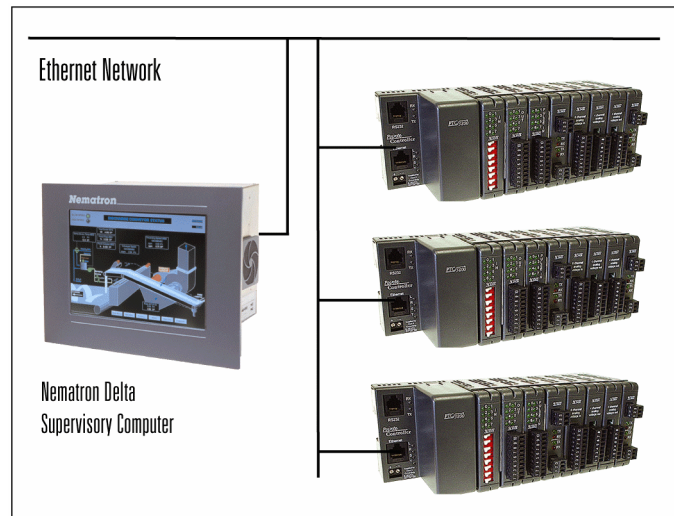
To configure additional OptiLogic RTUs:

1. Launch the PointeControl Framework and open your project.
2. From the **Project** menu, choose **Configure I/O**.
3. Click the **RTU 1** tab.
4. Click the **Base type** drop-down menu and select the appropriate OptiLogic RTU model number: OL4054, OL4058, or OL4228.
5. In the **Unit identifier** field, enter the RTU's ID number.
6. Specify what I/O modules and operator panel are installed in the RTU. This is done the same way as for the Pointe Controller itself. For a reminder, see "[Specifying your installed hardware](#)" on page 121.
7. Configure the I/O modules. This is done the same way as for the Pointe Controller itself. For a reminder, see "[Configuring I/O modules](#)" on page 127.
8. Configure the operator panel, if any. This is done the same way as for the Pointe Controller itself. For a reminder, see "[Configuring operator panels](#)" on page 128.
9. Repeat steps 2 through 7 for **RTU 2**, **RTU 3**, and **RTU 4** as needed.
10. Click **OK** to save your changes and close the window.

For more information on using OptiLogic RTUs, see the *OptiLogic RTU User Manual*. This document is included with every OptiLogic RTU, or it can be downloaded from Optimization's Web site at <http://www.optimize.com/>.

8.2 Networking via Modbus Data Mapping

As a control node in a scalable network, the Pointe Controller performs dedicated real-time local control, while maintaining communications with the designated supervisory computer. Total system deployment, configuration, project coordination, and data logging can be implemented from any authorized network workstation.



The Pointe Controller can be configured to communicate with supervisory computers and operator terminals via the industry-standard Modbus protocol. To do this, you must first enable the Modbus driver and then map the **Logic Memory** variables used in PointeControl to the appropriate Modbus addresses. When your project is downloaded to and run on the Pointe Controller unit, Modbus communication is started automatically and the unit shares the mapped data over its serial and/or Ethernet connection.

If you intend to configure Modbus mapping, you should be familiar with the Modbus protocol and the different **Modbus data types**, as defined by Modicon / Schneider Electric. You should also have a basic understanding of PLC memory addressing.

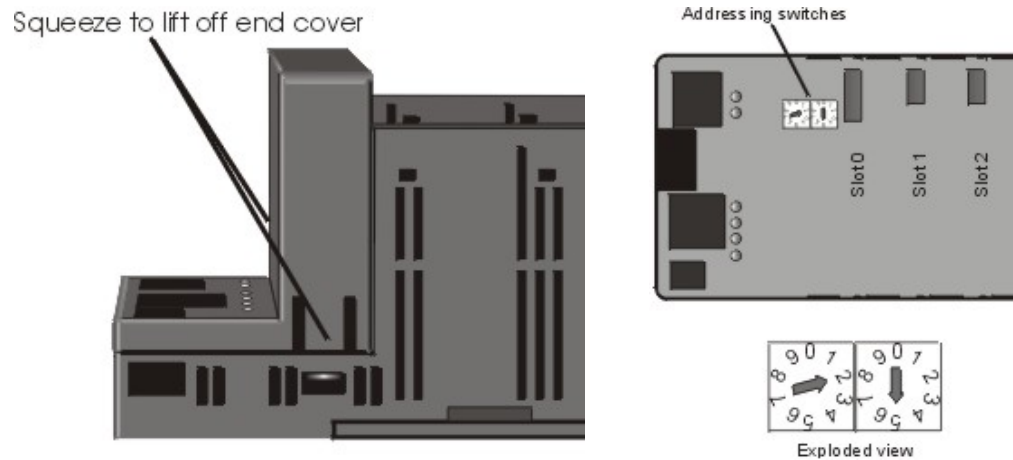
NOTE: The Pointe Controller is designed to work as a Modbus slave device, regardless of whether serial or Ethernet communication is enabled. Anything that attaches to the Pointe Controller unit via Ethernet is automatically the Modbus master device and therefore responsible for initiating requests to the controller.

8.2.1 Modbus Address

The addressing that you, the system designer, must set is the address set via rotary address switches in the Pointe Controller base unit. Every Modbus device in your system must have its own unique address. This address, a value between 00 and 97, is how the Modbus master identifies each device.

NOTE: Addresses 98 and 99 are reserved for performing hardware resets. For more information, see page 230.

To get to the address switches, you must first remove the end cover from the base unit. To do this, simply squeeze the latching tabs, shown in the figure below, and lift the cover off.



Removing the end cover will expose the base motherboard. The address switches will be found near the connector for slot 0.

To set the controller’s Modbus address, rotate the switches to the desired values. The switch on the left is the “tens” digit. The switch on the right is the “ones” digit. A small flat blade screwdriver is the only tool you need. The address shown on the figure above is “25.”

Remember that each Pointe Controller unit on your network must have its own unique Modbus address, which is set prior to applying power to the controller. Duplicate addresses will cause system communications to fail.

8.2.2 Types of Modbus data

PointeControl allows mapping to four different Modbus data types: Coils, Discretes, Analogs, and Registers. The following table shows which variables in Logic Memory can be mapped to each data type:

	Coils (00001-09999) read/write	Discretes (10001-19999) read only	Analogs (30001-39999) read only	Registers (40001-49999) read/write
Inputs				
%IX (Bits)		X		
%IUB (8 Bit Unsigned)			X	
%IB (8 Bit Signed)			X	
%IUW (16 Bit Unsigned)			X	
%IW (16 Bit Signed)			X	
%IUD (32 Bit Unsigned)			X	
%ID (32 Bit Signed)			X	
%IF (32 Bit Real)			X	
Memory				
%MX (Bits)	X	X		X
%MUB (8 Bit Unsigned)			X	X
%MB (8 Bit Signed)			X	X
%MUW (16 Bit Unsigned)			X	X
%MW (16 Bit Signed)			X	X
%MUD (32 Bit Unsigned)			X	X
%MD (32 Bit Signed)			X	X
%MF (32 Bit Real)			X	X
Outputs				
%QX (Bits)	X	X		X
%QUB (8 Bit Unsigned)			X	X
%QB (8 Bit Signed)			X	X
%QUW (16 Bit Unsigned)			X	X
%QW (16 Bit Signed)			X	X
%QUD (32 Bit Unsigned)			X	X

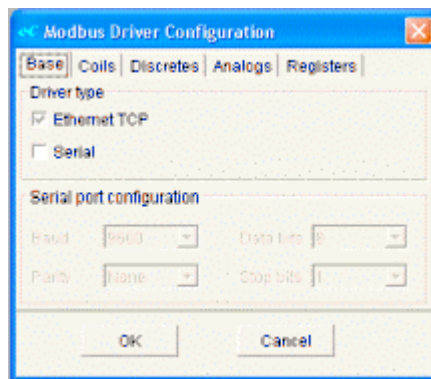
	Coils (00001-09999) read/write	Discretes (10001-19999) read only	Analogs (30001-39999) read only	Registers (40001-49999) read/write
%QD (32 Bit Signed)			X	X
%QF (32 Bit Real)			X	X
Strings			X	X
Timers			X	

WARNING: Always be careful when mapping directly to Output variables. While it may be desirable to do so in certain applications, be advised that if you do, you will be giving direct control of mapped outputs to any remote device that communicates over your Modbus interface. It is generally more advisable to map to a Memory variable, so that your PointeControl program can check incoming commands and verify they should be executed.

8.2.3 Enabling the Modbus driver

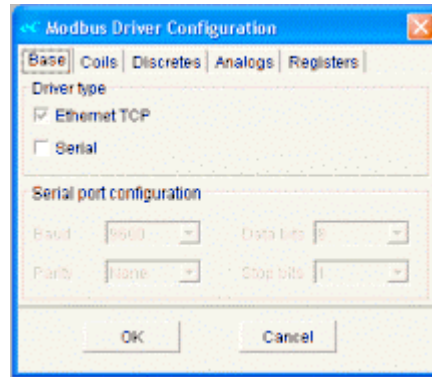
To enable the Modbus driver on the Pointe Controller unit:

1. Launch the PointeControl Framework and open your project.
2. From the **Project** menu, choose **Configure Modbus Mapping**. The Modbus Driver Configuration window will appear.



3. To enable Modbus communication via Ethernet, click the **Ethernet TCP** checkbox. No additional configuration is needed.

4. To enable Modbus communication via the RS232 serial port:
 - a. Click the **Serial** checkbox. The **Serial port configuration** settings become enabled.



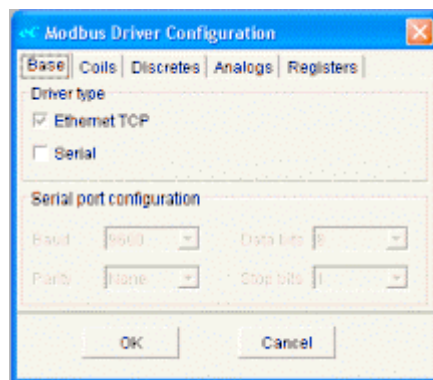
- b. Configure the serial port as needed for your control application. (The default values of 9600 baud, 8 data bits, 1 stop bit and no parity are recommended.)
5. Proceed to [Mapping variables to Modbus addresses](#), or click **OK** to save your changes and close the Modbus Driver Configuration window.

NOTE: You can enable both Ethernet and Serial communications on the same Pointe Controller unit. However, each type of communications requires additional memory and processing power, so enable only the features you need for your control application.

8.2.4 Mapping variables to Modbus addresses

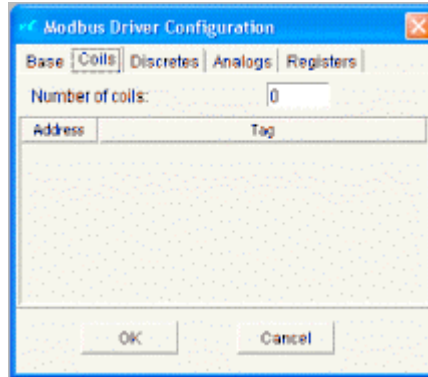
To map Logic Memory variables to Modbus addresses:

1. If the Modbus Driver Configuration window is not already open, open it now by choosing **Project > Configure Modbus Mapping**.



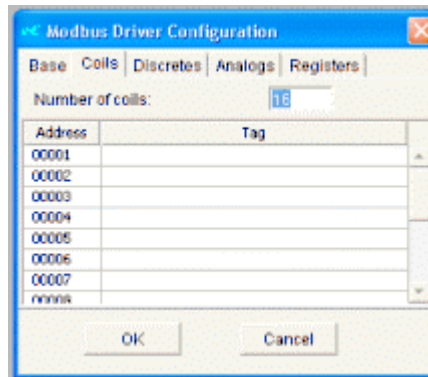
- Click on the tab corresponding to the Modbus data type — **Coils**, **Discretes**, **Analogs**, or **Registers** — that you want to map.

In this example, **Coils** is selected:



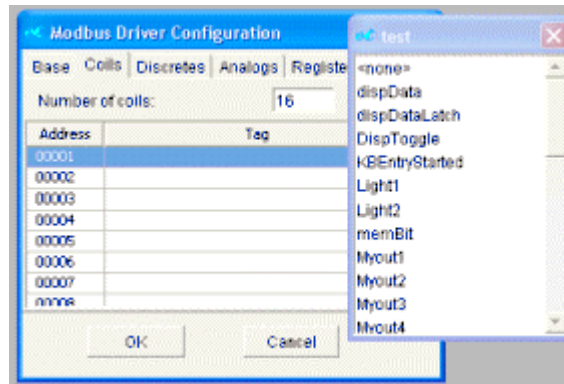
- Specify the number of addresses that you want to map in this Modbus data type: click in the **Number of** field, enter the number, and press the Tab key. Addressing always starts at the low end of the available range.

In this example, 16 addresses are specified, numbered 00001 through 00016:



NOTE: You can increase the number of addresses at any time without affecting addresses that have already been mapped. However, if you attempt to *decrease* the number of addresses without first unmapping the addresses that would be removed, you will be prompted to verify the action.

- Double-click on an address you want to map. A pop-up window appears listing all of the currently defined Logic Memory variables that can be mapped to that Modbus data type.



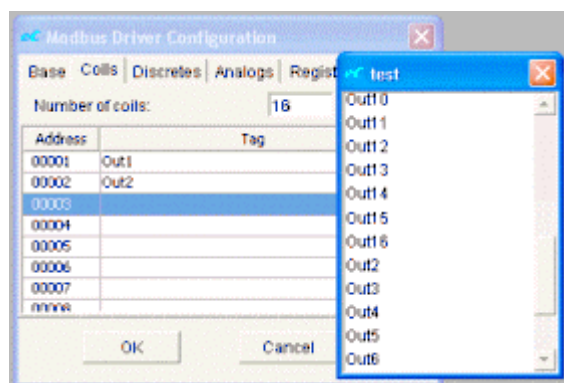
For more information on which Logic Memory variables can be mapped to which Modbus data types, see [“Types of Modbus data”](#) on page 219.

- Select the variable you want to map from the pop-up window. The variable is mapped to the address.

NOTE: Each Modbus address represents a 16-bit memory location. If the selected variable — for example, a string or a long integer — requires more than 16 bits, then additional addresses will be allocated to the variable and marked as <unavailable>.

Also, PointeControl will automatically append a NULL terminator onto a string as it is mapped. Therefore, as an example, a 20-character string will be allocated 11 addresses: 10 addresses for the string at two characters per address, plus one address for the NULL terminator.

- Continue mapping each address until finished.



NOTE: You do not need to map all of the addresses in a data type. You can leave some blank while mapping only the addresses needed to communicate with your other Modbus devices.

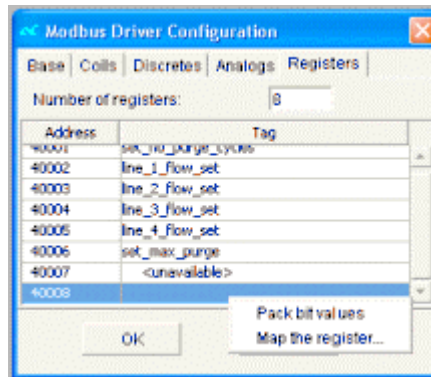
- Return to step 2 to map another data type, or click **OK** to save your changes and close the Modbus Driver Configuration window.

Packing individual bits into a Register

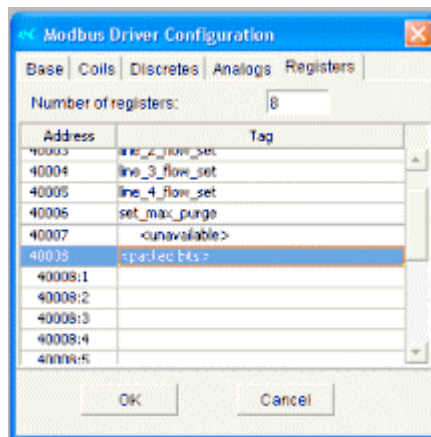
Under the Register data type, you can “pack up to 16 individual Bit-type variables into a single address. This is done by expanding the address into 16 sub-addresses and mapping the bits to each sub-address. Each sub-address is denoted by a “:n suffix on the address.

To pack individual bits into a Register:

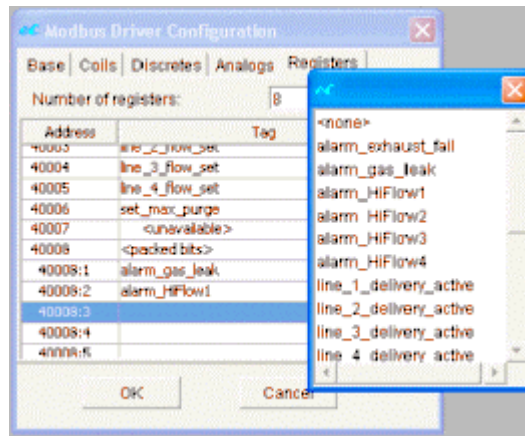
- Click the **Registers** tab.
- Right-click on the Register address you want to map and choose **Pack bit values** from the menu.



The address is expanded into 16 sub-addresses.



- Double-click on the chosen sub-address and select the bit to be mapped from the pop-up window.



- Continue mapping each sub-address until finished.

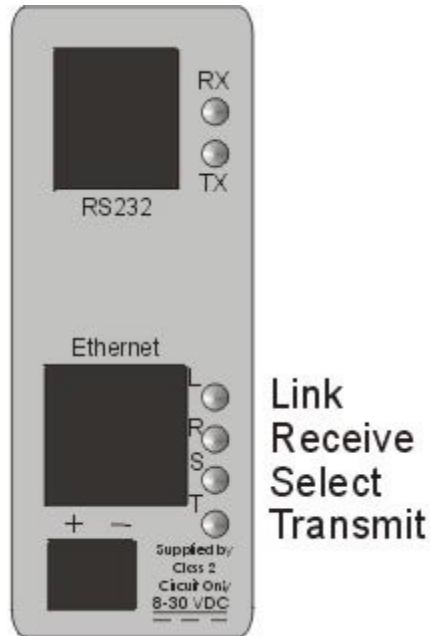
To unpack a Register – that is, to delete the individual mappings and convert the Register back to a full 16-bit address – right-click on the Register address and choose **Undo packing** from the menu.

Chapter 9: Troubleshooting

This chapter provides some basic tips for troubleshooting the Pointe Controller system. More information will be added in future revisions. In the meantime, if you have any problems, please contact Nematron Technical Support at 1-800-636-2876, or email us at support@nematron.com.

9.1 LED Boot Indicators

The first thing to look at when the controller is installed, connected to a hub or PC and everything is powered up, is the diagnostic LEDs:



The first thing that happens when the controller is powered up is that it checks its operating program. This process takes a couple of seconds. If the operating program does not check out, the RS232 TX (transmit) LED will flash at a rate of about 1 flash per second. If this should happen, the base must be loaded with operating software.

After the startup program check (as long as a programming cable is not plugged into the RS232 port), the base will enter its main program. At this point, the Select LED should be on indicating the program is interfacing the ethernet electronics.

The next thing to look at is the Link (L) LED. If it is on, there is a good ethernet link. Ethernet devices send a periodic "link pulse". The ethernet receiver on the other side looks for this link signal. If it is received, the link LED will light. Link LEDs should be on, both on the RTU and the hub.

If the link LEDs do not come on, one of the following problems probably exists.

- The cable between the hub and the RTU is defective (improper connections, bad connections, etc.)
- The hub or the RTU is not turned on.

There are two LEDs, one red and one green, next to the Pointe Controller base unit's RJ-11 connector. The red LED is used by the boot procedure to indicate failure conditions:

Flashes	Failure Condition
1	Flash Memory test failed
2	RAM test failed
3	LAN Controller Register test failed
4	Missing memory size configuration
5	Serial port failed to initialize

If any of the above failure conditions exist, then the red LED will continue to flash the corresponding number of times with a short pause between flash cycles. For example, if the RAM test has failed, then the red LED will flash three times, pause, flash three times, pause, flash three times, and so on.

NOTE: In most cases, a failure condition may be cleared by cycling power to the Pointe Controller unit. If the failure condition reoccurs, then contact Nematron customer support.

If all boot tests pass and no failure condition exists, then the unit checks to see if the serial download cable is connected. If it is, then the unit waits for download commands from the configuration utility (Update Tool).

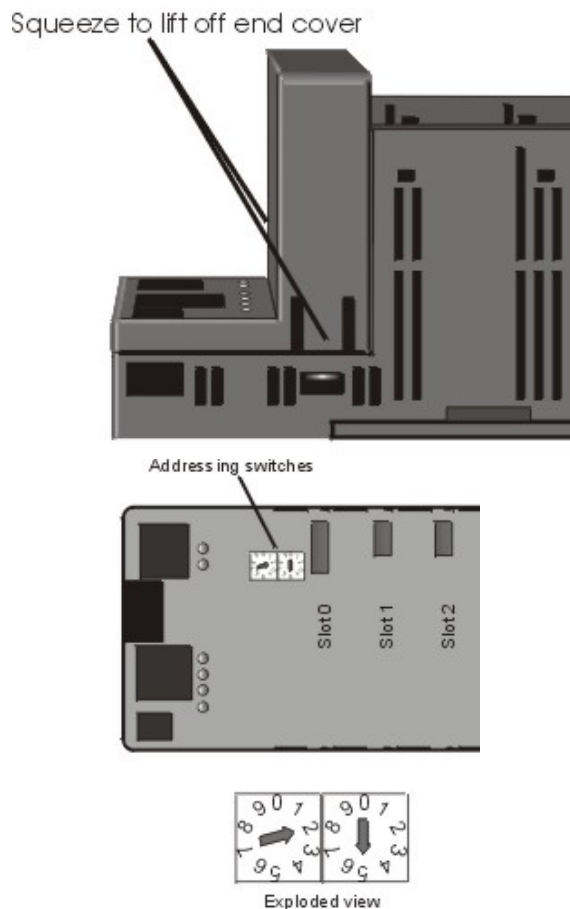
If the serial download cable is not connected, then the unit will attempt to run the project currently stored in memory, if any. If there is no project in memory or if the project fails to run, then the red LED flashes rapidly without pauses to indicate that user action is required.

9.2 Hardware Reset

In some cases, it may be necessary to perform a full hardware reset on the Pointe Controller unit. This will bypass any password set on the controller, reset the IP address and node name settings to factory defaults, and erase any control program currently saved in flash memory.

The reset is performed by powering off the controller, turning the Modbus address switches to "99," and powering on the controller. Upon startup, the controller will detect the new address and immediately reset itself. After that, you can reconfigure the controller as if it was brand new. (See Chapter 2, "Initial Setup," starting on page 22.)

To get to the address switches, you must first remove the end cover from the base unit. To do this, simply squeeze the latching tabs, shown in the figure below, and lift the cover off.



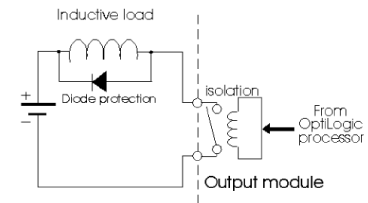
Removing the end cover will expose the base motherboard. The address switches will be found near the connector for slot 0. To set the address, rotate both switches to the "9" position. A small flat blade screwdriver is the only tool you need.

Appendix A: OptiLogic Technical Specifications

This appendix provides complete technical descriptions and configuration instructions for all of the OptiLogic I/O modules and operator panels that can be used with the Pointe Controller system.

A.1 OL2104 Relay Output Module

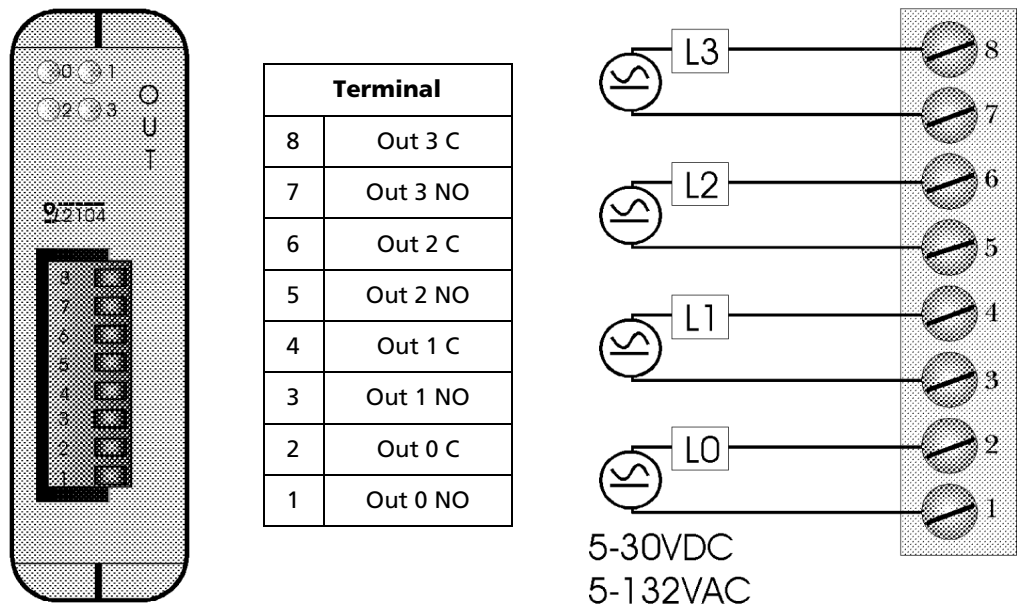
The OL2104 Relay Output Module provides four (4) optically isolated mechanical relay outputs that can be used for switching a variety of AC and DC loads. Individual LED indicators provide visual feedback of output state.



Technical Specifications

Card Cage Power Required	250 mA
Output Type	Mechanical relay
Outputs	4
Status Indicators	Logic Side LED
Contact resistance	0.1 ohm (initial)
Contact voltage rating	0 - 60 VDC 0 - 120 VAC
Contact rating	2A (resistive) / point @24 VDC, 1A / point @120 VAC
Minimum load	10 mA
Contact type	Form A (SPST)
Contact arrangement	4 isolated normally open contact relays
Mechanical life	10,000,000 operations per relay (at no load)
Electrical life	100,000 operations per relay (at full load)
Terminal Strip	Plug In (removable)
Terminal Screws	Slotted (0.1" blade max)
Max. terminal wire gauge	18 AWG (use copper conductors)
Terminal block torque	2.2 lb-in
Required Temp. rating of field installed conductors	60°C / 75°C
Weight	1.6 oz (46 g)
Type	8
Subtype	1

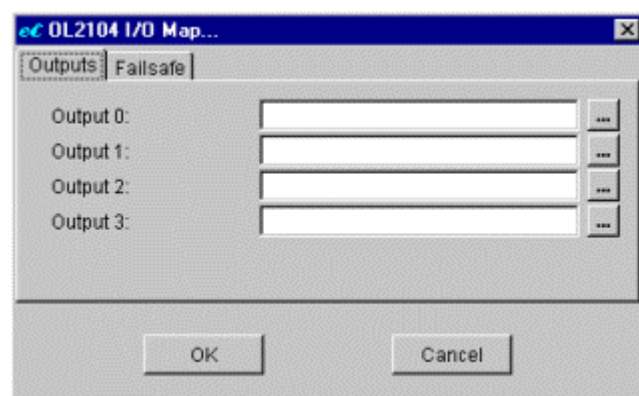
Connection Diagram



A.1.1 OL2104 Configuration Options

The OL2104 module is configured through the Configure I/O menu command in the PointeControl development framework. (For more information on Configure I/O, see page 118.) Selecting an OL2104 module and clicking the **I/O** button opens the OL2104 I/O Map dialog window...

Outputs tab

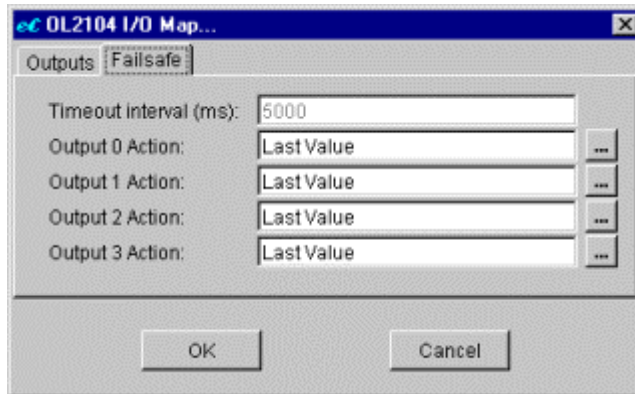


Each **Output** point, from 0 through 3, is associated with a single Output Bit tag (%QX).

To configure an Output, click the **...** to the right and select an Output Bit tag from the pop-up menu. Each tag can be used only once.

NOTE: You can leave any or all Outputs unconfigured, but if you do, those I/O points will not be available to your project.

Failsafe tab




If, during runtime, your project fails to respond for a given timeout interval, then the Pointe Controller unit will automatically set all Output points to preconfigured failsafes. Parameters for this tab include:

- **Timeout Interval (ms)** – Wait time in milliseconds after project failure before Outputs switch to configured Output Actions.

NOTE: This value of Timeout Interval is taken from the project's preferences and is not editable in this window. To edit this value, see [Edit Preferences](#).

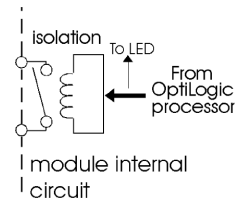
- **Output *n* Action** – Each Output has three possible failsafe actions:
 - Fail ON – On Fail condition, turn this Output OFF.
 - Fail OFF – On Fail condition, turn this Output ON.
 - Last Value (default) – On Fail condition, leave this Output in its current state.

To configure an Output Action, click the  to the right and select an action from the pop-up menu.

When you have finished configuring the module, click **OK** to save your changes and close the window.

A.2 OL2108 Relay Output Module

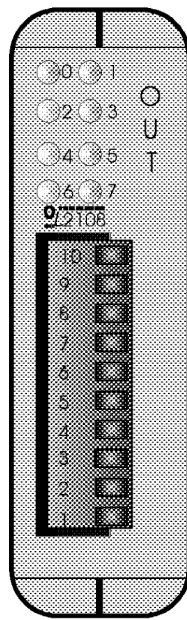
The OL2108 Relay Output Module provides eight (8) optically isolated mechanical relay outputs that can be used for switching a variety of AC and DC loads. Individual LED indicators provide visual feedback of output state.



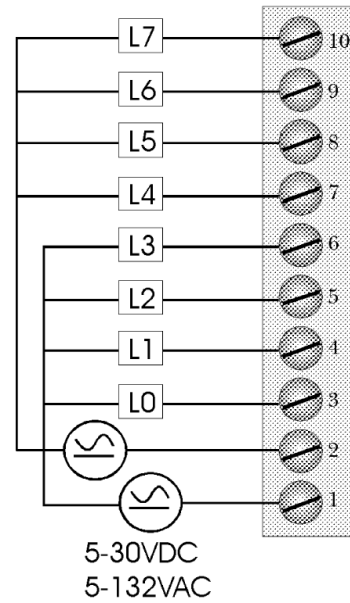
Technical Specifications

Card Cage Power Required	375 mA
Output Type	Mechanical relay
Outputs	8
Status Indicators	Logic Side LED
Contact resistance	0.1 ohm (initial)
Contact voltage rating	0 - 60 VDC 0 - 120 VAC
Contact rating	2A (resistive) / point @24 VDC, 1A / point @120 VAC
Minimum load	10 mA
Contact type	Form A (SPST)
Contact arrangement	4 relays per common
Mechanical life	10,000,000 operations per relay (at no load)
Electrical life	100,000 operations per relay (at full load)
Terminal Strip	Plug In (removable)
Terminal Screws	Slotted (0.1" blade max)
Max. terminal wire gauge	18 AWG (use copper conductors)
Terminal block torque	2.2 lb-in
Required Temp. rating of field installed conductors	60°C / 75°C
Weight	2.1 oz (58 g)
Type	9
Subtype	1

Connection Diagram



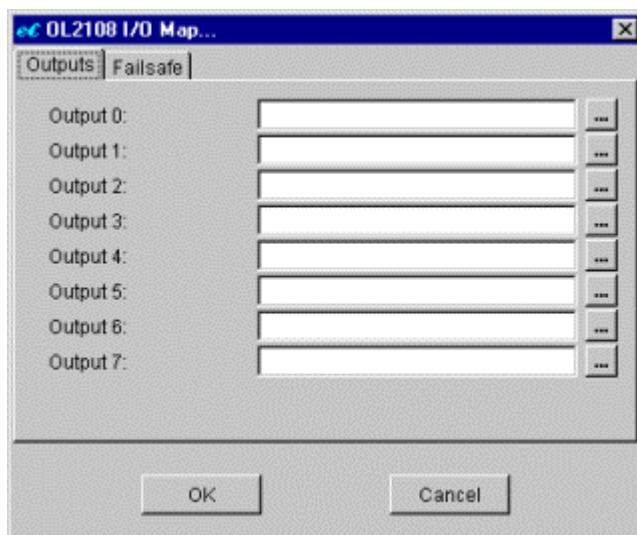
Terminal	
10	Out 7
9	Out 6
8	Out 5
7	Out 4
6	Out 3
5	Out 2
4	Out 1
3	Out 0
2	Common Out 4-7
1	Common Out 0-3




A.2.1 OL2108 Configuration Options

The OL2108 module is configured through the Configure I/O menu command in the PointeControl development framework. (For more information on Configure I/O, see page 118.) Selecting an OL2108 module and clicking the **I/O** button opens the OL2108 I/O Map dialog window...

Outputs tab

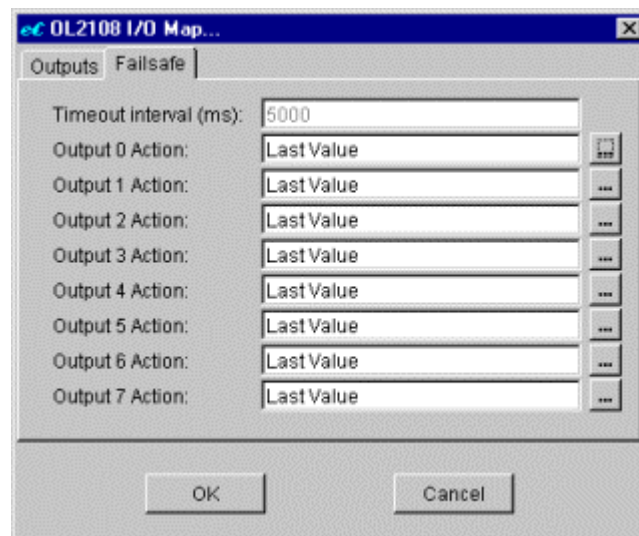


Each **Output** point, from 0 through 7, is associated with a single Output Bit tag (%QX).

To configure an Output, click the  to the right and select an Output Bit tag from the pop-up menu. Each tag can be used only once.

NOTE: You can leave any or all Outputs unconfigured, but if you do, those I/O points will not be available to your project.

Failsafe tab




If, during runtime, your project fails to respond for a given timeout interval, then the Pointe Controller unit will automatically set all Output points to preconfigured failsafes. Parameters for this tab include:

- **Timeout Interval (ms)** – Wait time in milliseconds after project failure before Outputs switch to configured Output Actions.

NOTE: This value of Timeout Interval is taken from the project’s preferences and is not editable in this window. To edit this value, see [Edit Preferences](#).

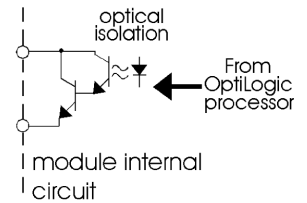
- **Output *n* Action** – Each Output has three possible failsafe actions:
 - Fail ON – On Fail condition, turn this Output OFF.
 - Fail OFF – On Fail condition, turn this Output ON.
 - Last Value (default) – On Fail condition, leave this Output in its current state.

To configure an Output Action, click the  to the right and select an action from the pop-up menu.

When you have finished configuring the module, click **OK** to save your changes and close the window.

A.3 OL2109 DC Sinking Output Module

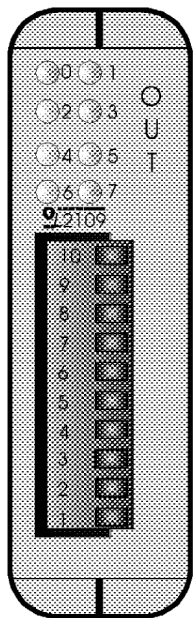
The OL2109 DC Sinking Output Module provides eight (8) optically isolated transistor outputs that can be used for switching small DC loads. Individual LED indicators provide visual feedback of output state.



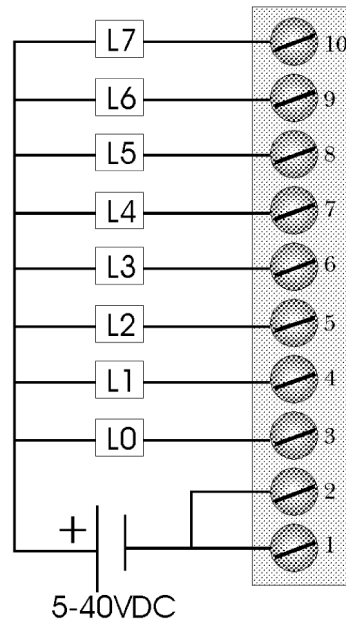
Technical Specifications

Card Cage Power Required	140 mA
Output Type	NPN open collector transistor
Outputs	8
Status Indicators	Logic Side LED
Voltage Rating	0 – 40VDC
Peak Voltage	80VDC
On voltage drop	0.75V @ 100mA 0.95V @ 300mA
Max. continuous load	300 mA
Maximum surge current	1.0A for 5 seconds
Commons	2 (connected internally)
Terminal Strip	Plug In (removable)
Terminal Screws	Slotted (0.1" blade max)
Max. terminal wire gauge	18 AWG (use copper conductors)
Terminal block torque	2.2 lb-in
Required Temp. rating of field installed conductors	60°C / 75°C
Weight	1.1 oz (30 g)
Type	9
Subtype	2

Connection Diagram



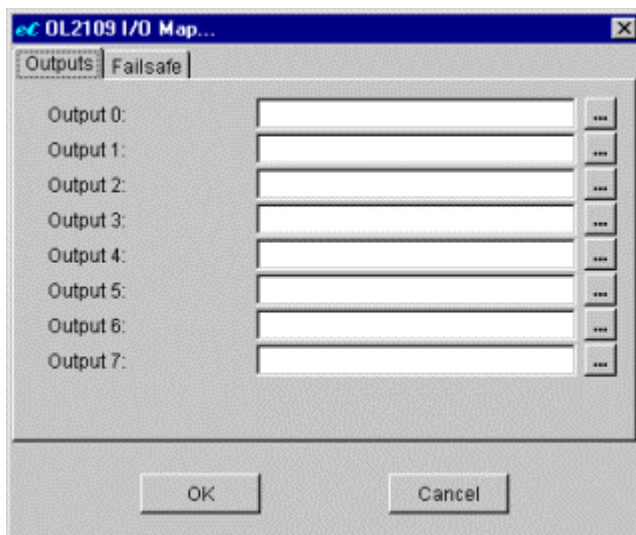
Terminal	
10	Out 7
9	Out 6
8	Out 5
7	Out 4
6	Out 3
5	Out 2
4	Out 1
3	Ou1 0
2	Common
1	Common




A.3.1 OL2109 Configuration Options

The OL2109 module is configured through the Configure I/O menu command in the PointeControl development framework. (For more information on Configure I/O, see page 118.) Selecting an OL2109 module and clicking the **I/O** button opens the OL2109 I/O Map dialog window...

Outputs tab

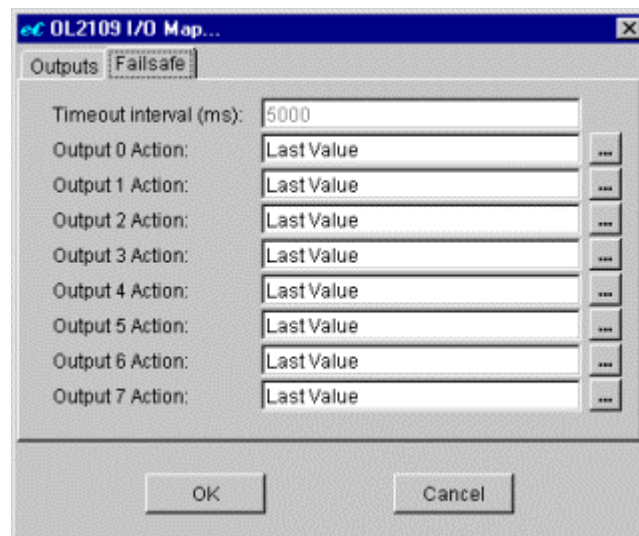


Each **Output** point, from 0 through 7, is associated with a single Output Bit tag (%QX).

To configure an Output, click the  to the right and select an Output Bit tag from the pop-up menu. Each tag can be used only once.

NOTE: You can leave any or all Outputs unconfigured, but if you do, those I/O points will not be available to your project.

Failsafe tab




If, during runtime, your project fails to respond for a given timeout interval, then the Pointe Controller unit will automatically set all Output points to preconfigured failsafes. Parameters for this tab include:

- **Timeout Interval (ms)** – Wait time in milliseconds after project failure before Outputs switch to configured Output Actions.

NOTE: This value of Timeout Interval is taken from the project’s preferences and is not editable in this window. To edit this value, see [Edit Preferences](#).

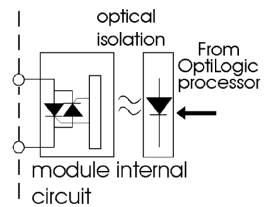
- **Output *n* Action** – Each Output has three possible failsafe actions:
 - Fail ON – On Fail condition, turn this Output OFF.
 - Fail OFF – On Fail condition, turn this Output ON.
 - Last Value (default) – On Fail condition, leave this Output in its current state.

To configure an Output Action, click the  to the right and select an action from the pop-up menu.

When you have finished configuring the module, click **OK** to save your changes and close the window.

A.4 OL2111 AC Solid State Relay Module

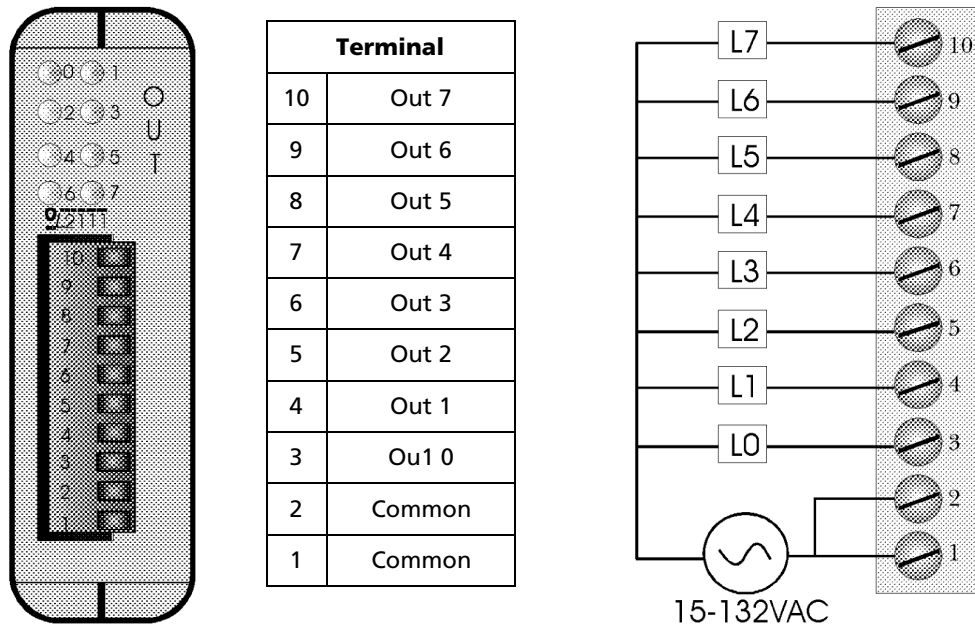
The OL2111 AC Solid-state Relay Module provides eight (8) solid-state relay outputs. This module is ideally suited for switching small AC loads. As a solid-state device, switch wear will not be a factor. Each output is optocoupled for system isolation. Individual LED indicators provide visual feedback indicating the state that each relay is being driven.



Technical Specifications

Card Cage Power Required	120 mA
Output Type	Solid state relay (Trice)
Outputs	8
Status Indicators	Logic Side LED
Voltage Rating	12 – 132 VAC
Maximum load current	0.5 A / point @ 120VAC
Minimum load current	10 mA
On state voltage drop	1V (typical)
Peak surge current, 1 cycle	15 A
Commons	2 (connected internally)
Terminal Strip	Plug In (removable)
Terminal Screws	Slotted (0.1" blade max)
Max. terminal wire gauge	18 AWG (use copper conductors)
Terminal block torque	2.2 lb-in
Required Temp. rating of field installed conductors	60°C / 75°C
Weight	1.3 oz (38 g)
Type	9
Subtype	3

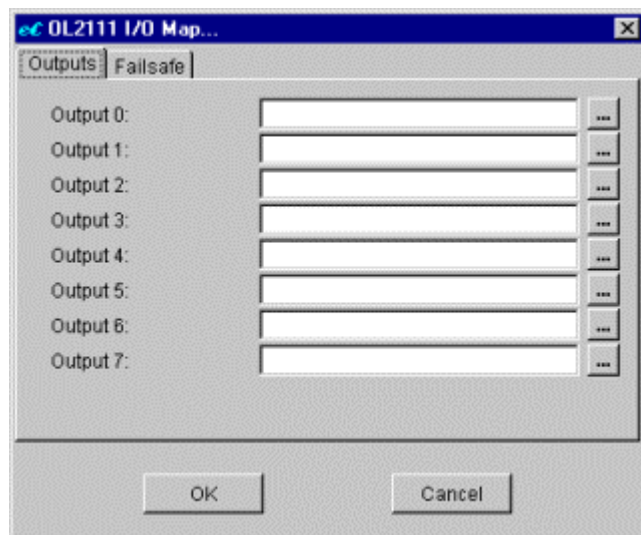
Connection Diagram




A.4.1 OL2111 Configuration Options

The OL2111 module is configured through the Configure I/O menu command in the PointeControl development framework. (For more information on Configure I/O, see page 118.) Selecting an OL2111 module and clicking the **I/O** button opens the OL2111 I/O Map dialog window...

Output tab

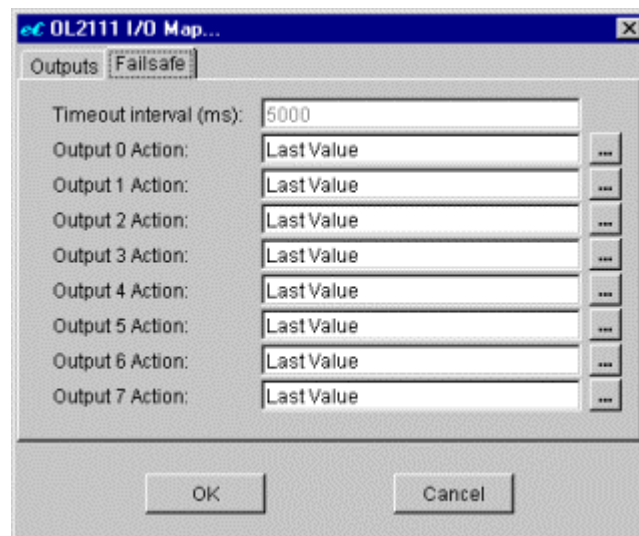


Each **Output** point, from 0 through 7, is associated with a single Output Bit tag (%QX).

To configure an Output, click the  to the right and select an Output Bit tag from the pop-up menu. Each tag can be used only once.

NOTE: You can leave any or all Outputs unconfigured, but if you do, those I/O points will not be available to your project.

Failsafe tab




If, during runtime, your project fails to respond for a given timeout interval, then the Pointe Controller unit will automatically set all Output points to preconfigured failsafes. Parameters for this tab include:

- **Timeout Interval (ms)** – Wait time in milliseconds after project failure before Outputs switch to configured Output Actions.

NOTE: This value of Timeout Interval is taken from the project’s preferences and is not editable in this window. To edit this value, see [Edit Preferences](#).

- **Output *n* Action** – Each Output has three possible failsafe actions:
 - Fail ON – On Fail condition, turn this Output OFF.
 - Fail OFF – On Fail condition, turn this Output ON.
 - Last Value (default) – On Fail condition, leave this Output in its current state.

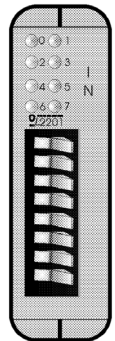
To configure an Output Action, click the  to the right and select an action from the pop-up menu.

When you have finished configuring the module, click **OK** to save your changes and close the window.

A.5 OL2201 Digital Input Simulator Module

The OL2201 Digital Input Simulator Module is designed to be an aid to program development. Use the OL2201 to simulate real world inputs during your design and debug process. The OL2201 enables the program developer to cause a change in input status at will to simulate a system action. In doing so, you are able to see the program's response. Use of the OL2201 is an aid in the process of thoroughly testing and debugging a system prior to "going live" with real hardware.

When it becomes time to move to real hardware, replace the OL2201 with the appropriate digital input module. The logic of your program will remain the same.



Technical Specifications

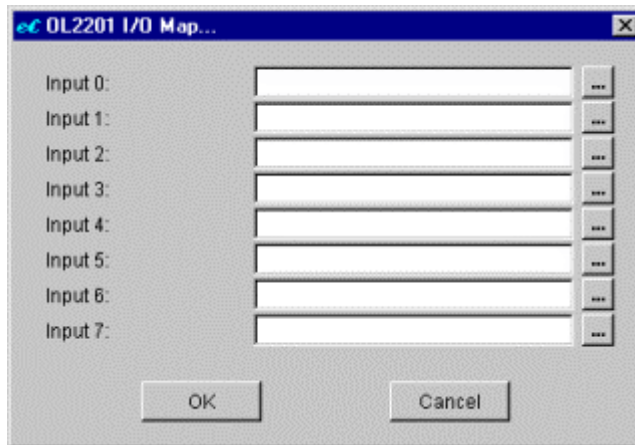
Card Cage Power Required	60mA
Input Type	Toggle Switch
Inputs	8
Status Indicators	Logic side LED
Weight	1.1 oz (30 g)
Type	1
Subtype	3

Connection Diagram


Because this module simulates inputs using toggle switches, there is no connection diagram.

A.5.1 OL2201 Configuration Options

The OL2201 module is configured through the Configure I/O menu command in the PointeControl development framework. (For more information on Configure I/O, see page 118.) Selecting an OL2201 module and clicking the **I/O** button opens the OL2201 I/O Map dialog window...



Configuring the OL2201 module is very simple: each **Input** point, from 0 through 7, is associated with a single Input Bit tag (%IX).

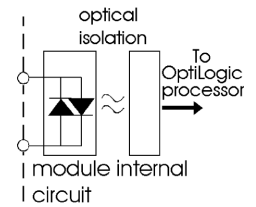
To configure an Input, click the  to the right and select an Input Bit tag from the pop-up menu. Each tag can be used only once.

NOTE: You can leave any or all Inputs unconfigured, but if you do, those I/O points will not be available to your project.

When you have finished configuring the module, click **OK** to save your changes and close the window.

A.6 OL2205 AC/DC Input Module

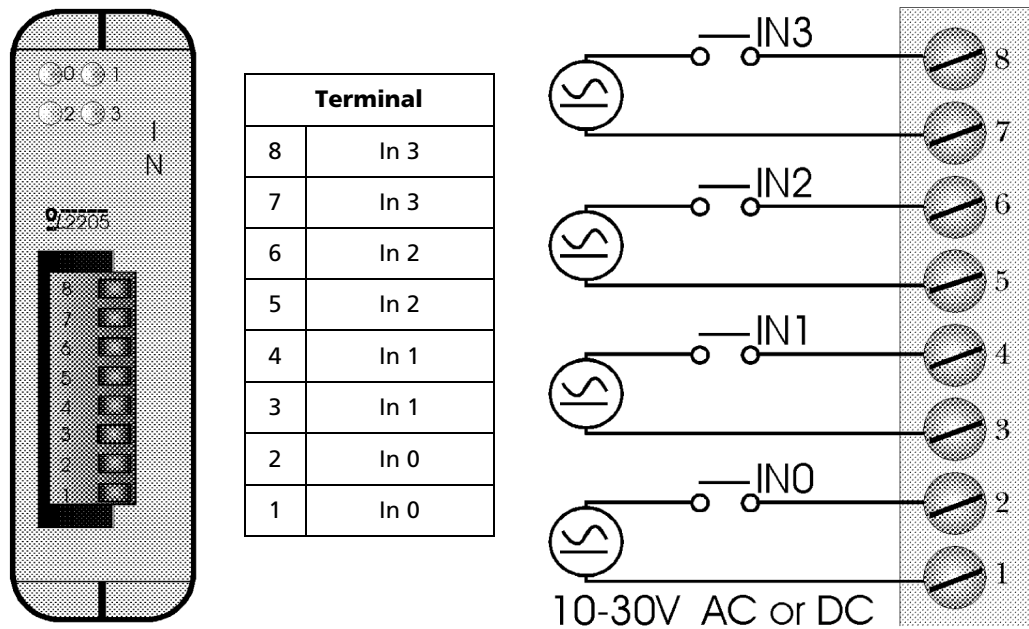
The OL2205 Digital Input module senses up to four (4) AC or DC input signals. All inputs are individually optocoupled for isolation. Inputs are also individually isolated from each other by separate terminal connections. Filtering is provided for zero crossover. Individual LED indicators provide visual feedback of current status.



Technical Specifications

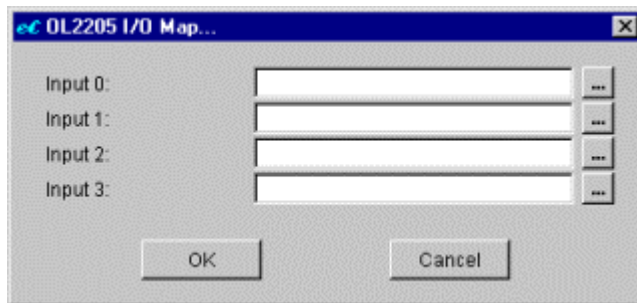
Card Cage Power Required	100 mA
Input Type	AC Optocoupled
Inputs	4
Status Indicators	Logic Side LED
Voltage Range	10-30 V AC or DC
Input Impedence	2.7K ohms
Inputs	DC sinking or sourcing / or AC
Min. On Current (per point)	3.3 mA
Max. On Current (per point)	11 mA
Terminal Strip	Plug In (removable)
Terminal Screws	Slotted (0.1" blade max.)
Max. terminal wire gauge	18 AWG (use copper conductors)
Terminal block torque	2.2 lb-in
Required Temp. rating of field installed conductors	60°C / 75°C
Weight	1.2 oz (34g)
Type	5
Subtype	1

Connection Diagram



A.6.1 OL2205 Configuration Options

The OL2205 module is configured through the Configure I/O menu command in the PointeControl development framework. (For more information on Configure I/O, see page 118.) Selecting an OL2205 module and clicking the **I/O** button opens the OL2205 I/O Map dialog window...



Configuring the OL2205 module is very simple: each **Input** point, from 0 through 3, is associated with a single Input Bit tag (%IX).

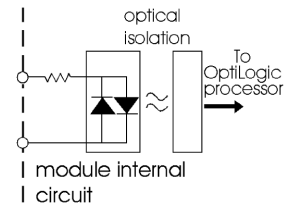
To configure an Input, click the **...** to the right and select an Input Bit tag from the pop-up menu. Each tag can be used only once.

NOTE: You can leave any or all Inputs unconfigured, but if you do, those I/O points will not be available to your project.

When you have finished configuring the module, click **OK** to save your changes and close the window.

A.7 OL2208 DC Digital Input Module

The OL2208 DC Digital Input module can be used in either sourcing or sinking application. (All eight inputs must be used in the same manner.) Each input is optocoupled to provide system isolation. Individual LED indicators provide a visual feedback of current status.

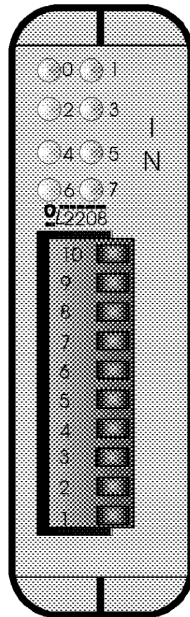


Technical Specifications

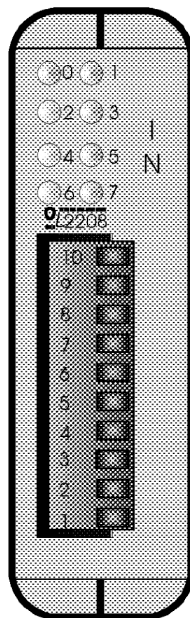
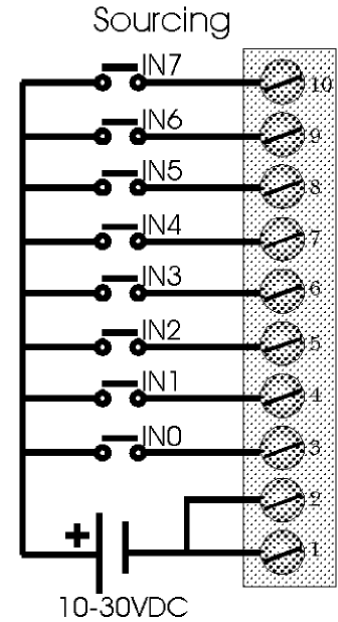
Card Cage Power Required	60 mA
Input Type	DC Optocoupled
Inputs	8
Status Indicators	Logic Side LED
Voltage Range	10 – 30 VDC
Input Impedence	2.7K ohms
Min. On Current (per point)	3.3 mA
Max. On Current (per point)	11 mA
Commons	2 (connected internally)
Terminal Strip	Plug In (removable)
Terminal Screws	Slotted (0.1" blade max.)
Max. terminal wire gauge	18 AWG (use copper conductors)
Terminal block torque	2.2 lb-in
Required Temp. rating of field installed conductors	60°C / 75°C
Weight	1.2 oz (34g)
Type	1
Subtype	1

Connection Diagram

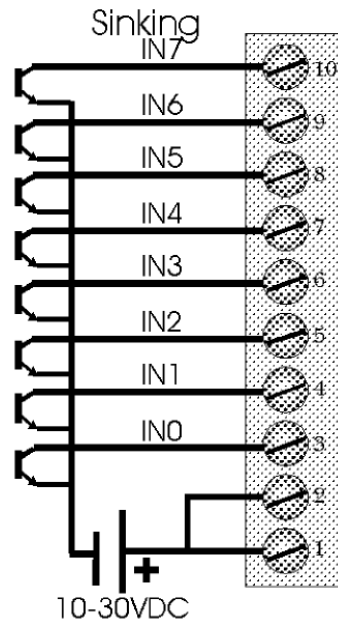
The OL2208 DC Digital Input module can be used in either sourcing or sinking application. All eight inputs must be used in the same manner.



Terminal	
10	In 7
9	In 6
8	In 5
7	In 4
6	In 3
5	In 2
4	In 1
3	In 0
2	Common
1	Common

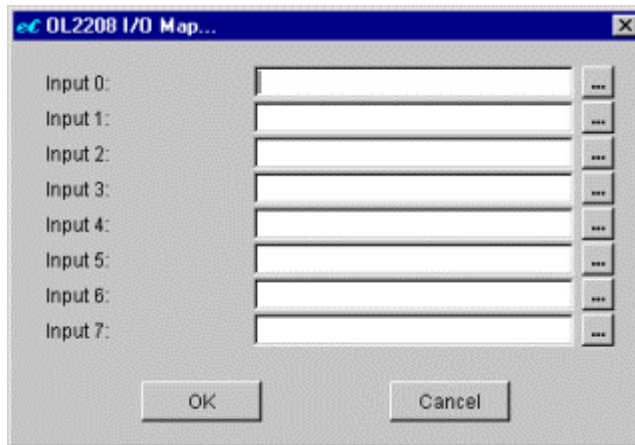


Terminal	
10	In 7
9	In 6
8	In 5
7	In 4
6	In 3
5	In 2
4	In 1
3	In 0
2	Common
1	Common




A.7.1 OL2208 Configuration Options

The OL2208 module is configured through the Configure I/O menu command in the PointeControl development framework. (For more information on Configure I/O, see page 118.) Selecting an OL2208 module and clicking the **I/O** button opens the OL2208 I/O Map dialog window...



Configuring the OL2208 module is very simple: each **Input** point, from 0 through 7, is associated with a single Input Bit tag (%IX).

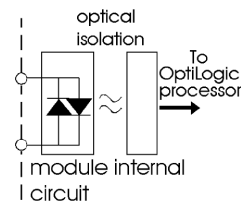
To configure an Input, click the  to the right and select an Input Bit tag from the pop-up menu. Each tag can be used only once.

NOTE: You can leave any or all Inputs unconfigured, but if you do, those I/O points will not be available to your project.

When you have finished configuring the module, click **OK** to save your changes and close the window.

A.8 OL2211 AC Digital Input Module

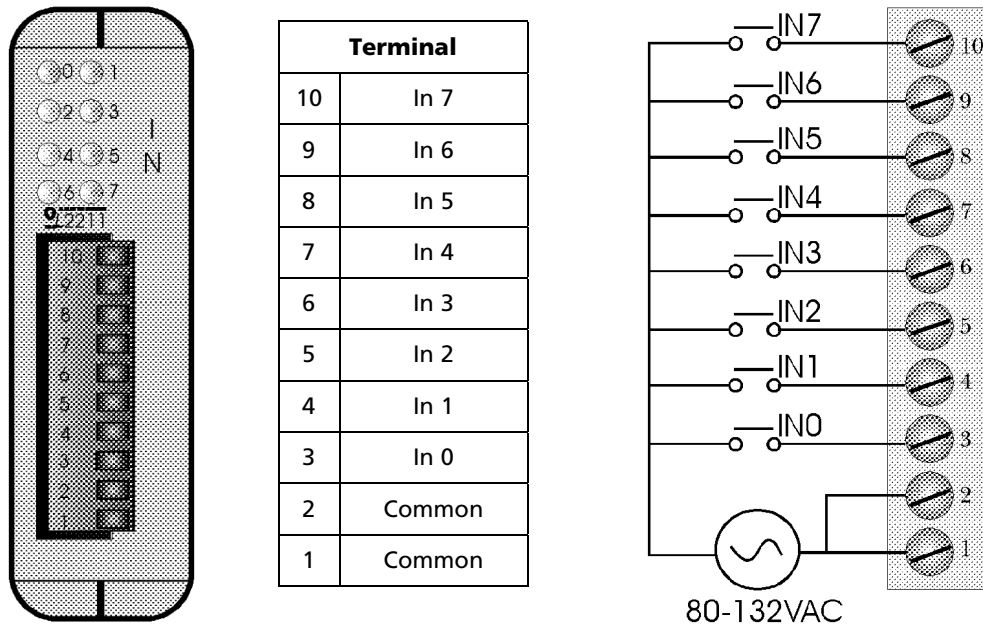
The OL2211 AC Digital input module senses up to eight (8) AC input signals. All inputs are individually optocoupled for isolation. Filtering is provided for zero crossover. Individual LED indicators provide visual feedback of current status.



Technical Specifications

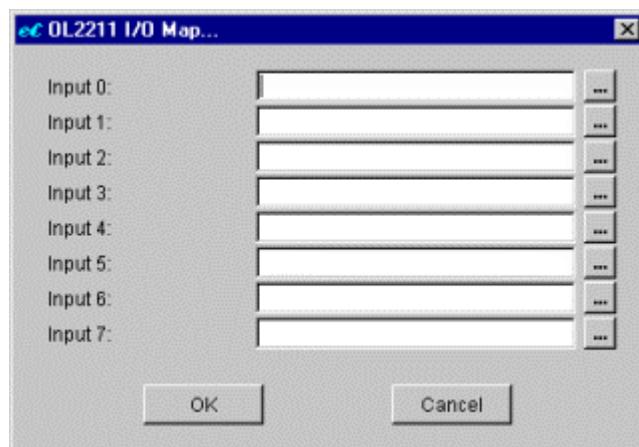
Card Cage Power Required	100 mA
Input Type	AC Optocoupled
Inputs	8
Status Indicators	Logic Side LED
Voltage Range	80 – 132 VAC
Input Impedance	47K ohms
Min. On Current (per point)	1.7 mA
Max. On Current (per point)	2.8 mA
Commons	2 (connected internally)
Terminal Strip	Plug In (removable)
Terminal Screws	Slotted (0.1" blade max.)
Max. terminal wire gauge	18 AWG (use copper conductors)
Terminal block torque	2.2 lb-in
Required Temp. rating of field installed conductors	60°C / 75°C
Weight	1.3 oz (38g)
Type	1
Subtype	2

Connection Diagram



A.8.1 OL2211 Configuration Options

The OL2211 module is configured through the Configure I/O menu command in the PointeControl development framework. (For more information on Configure I/O, see page 118.) Selecting an OL2211 module and clicking the **I/O** button opens the OL2211 I/O Map dialog window...



Configuring the OL2211 module is very simple: each **Input** point, from 0 through 7, is associated with a single Input Bit tag (%IX).

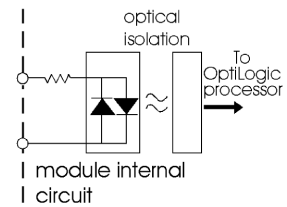
To configure an Input, click the **...** to the right and select an Input Bit tag from the pop-up menu. Each tag can be used only once.

NOTE: You can leave any or all Inputs unconfigured, but if you do, those I/O points will not be available to your project.

When you have finished configuring the module, click **OK** to save your changes and close the window.

A.9 OL2252 Dual Pulse Counter

The OL2252 module provides two independent high speed pulse counter inputs. Each input counter will accurately count pulse inputs up to 15KHz. Inputs may be sourcing or sinking type. There are a number of operating options available with the OL2252. The six remaining inputs can be used as predefined control signals or as general purpose inputs. These options are detailed below.



Input Connections

The following is a list of the input connections on the module:

Terminal	Label	Description
1	Common	Sourcing or sinking return line
2	Common	Sourcing or sinking return line
3	Pulse 1	Square wave input, up to 15 KHz
4	Pulse 2	Square wave input, up to 15 KHz
5	Reset 1	If configured as "reset" input, will clear the Pulse 1 count when activated. If not configured as "reset" input, can be used as a general purpose input.
6	Reset 2	If configured as "reset" input, will clear the Pulse 2 count when activated. If not configured as "reset" input, can be used as a general purpose input.
7	Enable 1	If configured as an "enable" input, enables the Pulse 1 counter when active. If not configured as an "enable" input, can be used as a general purpose input.
8	Enable 2	If configured as an "enable" input, enables the Pulse 2 counter when active. If not configured as an "enable" input, can be used as a general purpose input.
9	Input 1	General purpose input
10	Input 2	General purpose input

Theory of Operation

The OL2252 Pulse Counter has two independent pulse counter inputs. These pulse counter inputs will accurately count pulses between 0 and 15KHz.

All counts begin at zero and count up to the maximum number the counter can hold (4,294,967,295). If the count should ever get that high, it will roll over to zero.

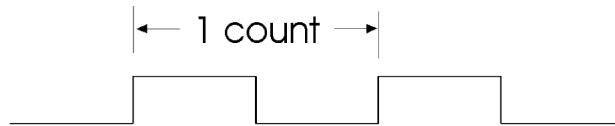
In order to count, the count input must be enabled. A message with an enable must come from the runtime program. The module can also be set up to use the local hardware input enable (in addition to the enable message).

The count can be reset to 0 at any time. Again there is both a reset message that can be sent from the PC and an optional hardware reset signal.

Whether the hardware “reset” and “enable” are used is determined by how the module is configured in the PointeControl development framework (see below).

Input Signal

The input pulse train is a repetitive square wave input that looks something like the following:



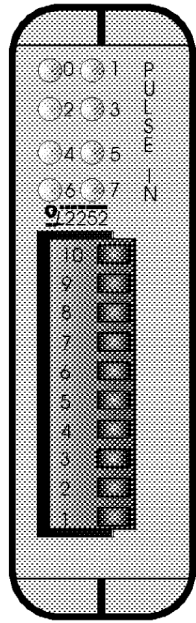
If you know the maximum frequency of the pulse train, you can configure the pulse counter to count pulse up to that pulse rate. In doing so, the counter will consider anything above the maximum rate that you have defined to be noise and will ignore it.

Technical Specifications

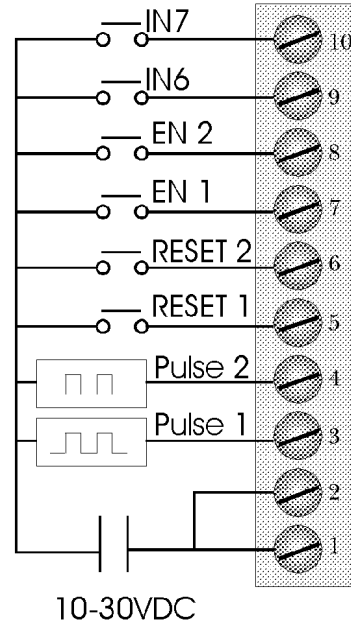
Card Cage Power Required	100 mA
Inputs (all)	8
Pulse Inputs	2
Status Indicators	Logic Side LED
Input Voltage	10 – 30 VDC
Input Impedence	2.7K ohms
Input frequency (on pulses)	15 KHz maximum
Min. On Current (per point)	3.3 mA
Max. On Current (per point)	11 mA
Commons	2 (connected internally)
Terminal Strip	Plug In (removable)
Terminal Screws	Slotted (0.1" blade max.)
Max. terminal wire gauge	18 AWG (use copper conductors)
Terminal block torque	2.2 lb-in
Required Temp. rating of field installed conductors	60°C / 75°C
Weight	1.2 oz (34g)

Type	1
Subtype	2

Connection Diagram



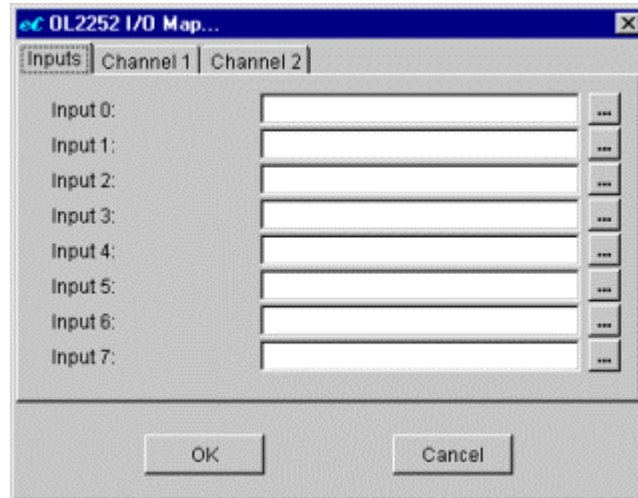
Terminal	
10	Input 2
9	Input 1
8	Enable 2
7	Enable 1
6	Reset 2
5	Reset 1
4	Pulse 2
3	Pulse 1
2	Common
1	Common




A.9.1 OL2252 Configuration Options

The OL2252 module is configured through the Configure I/O menu command in the PointeControl development framework. (For more information on Configure I/O, see page 118.) Selecting an OL2252 module and clicking the **I/O** button opens the OL2252 I/O Map dialog window...

Inputs tab

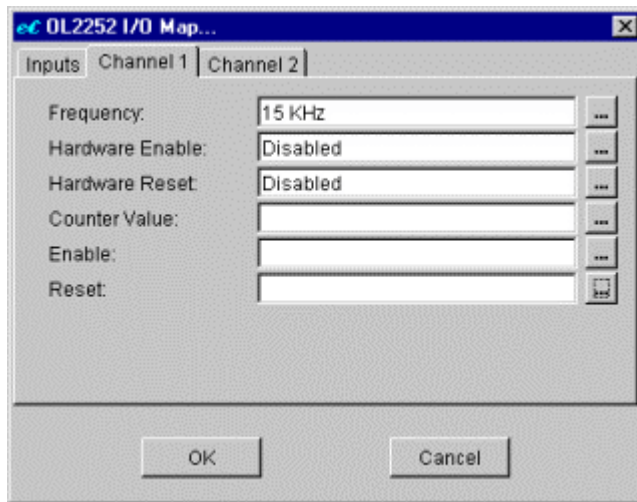


Each **Input** point, from 0 through 7, is associated with a single Input Bit tag (%IX).

To configure an Input, click the  to the right and select an Input Bit tag from the pop-up menu. Each tag can be used only once.

NOTE: You can leave any or all Inputs unconfigured, but if you do, the pulse counters may not work as intended.

Channel tabs




The first six Inputs on the OL2252 module are grouped into two channels, which correspond with and control the two pulse counters on the module. (The last two Inputs – 6 and 7 – are generic DC inputs.) The Inputs are grouped as follows:


Channel	Counter Input	Hardware Reset	Hardware Enable
1	Input 0	Input 2	Input 4
2	Input 1	Input 3	Input 5

These channels are configured via the Channel 1 and Channel 2 tabs. Both tabs have the same parameters:

- **Frequency** – The maximum frequency range for the channel to count pulses. Available frequencies: 15 KHz (default), 10 KHz, 5 KHz, 2.5 KHz, 1 KHz.


To configure Frequency, click the  to the right and select a frequency from the pop-up menu.

- **Hardware Enable** – Option to use value received on the Hardware Enable input (Input 4 for Channel 1, Input 5 for Channel 2) to determine when the channel starts and stops counting.
 - If Disabled (default), then the Hardware Enable input can be used as a generic DC input and will not have any effect on the channel.
 - If Enabled, then the value received on the Hardware Enable input controls the corresponding channel’s behavior: a value of ON starts counting, while a value of OFF stops counting.


To configure Hardware Enabled, click the  to the right and select an option (Disabled or Enabled) from the pop-up menu.

NOTE: If you use the Hardware Enable option, then you must also have the Enable bit set (see below). If you do not, the channel cannot count.


- **Hardware Reset** – Option to use value received on the Hardware Reset input (Input 2 for Channel 1, Input 3 for Channel 2) to determine when the channel resets the counter to 0.
 - If Disabled (default), then the Hardware Reset input can be used as a generic DC input and will not have any effect on the channel.
 - If Enabled, then the value received on the Hardware Reset input controls the corresponding channel's behavior: a value of ON resets the counter to 0, while a value of OFF resumes counting as normal.

To configure Hardware Reset, click the  to the right and select an option (Disabled or Enabled) from the pop-up menu.


- **Counter Value** – The 32-bit Unsigned Input tag (%IUD) to which the channel's actual counter total is written.

To configure Counter Value, click the  to the right and select a tag from the pop-up menu. Each tag can be used only once.

- **Enable** – Bit to enable and disable the counter from within the project. This bit takes precedence over the Hardware Enable input (above). When the bit is ON, the counter is enabled. When the bit is OFF, the counter is disabled. Mapped to an Output Bit tag (%QX).

To configure Enable, click the  to the right and select a tag from the pop-up menu. Each tag can be used only once.

- **Reset** – Bit to reset and resume the counter from within the project. This bit takes precedence over the Hardware Reset input (above). When the bit is ON, the counter is reset. When the bit is OFF, the counter is resumed. Mapped to an Output Bit tag (%QX).

To configure Reset, click the  to the right and select a tag from the pop-up menu. Each tag can be used only once.

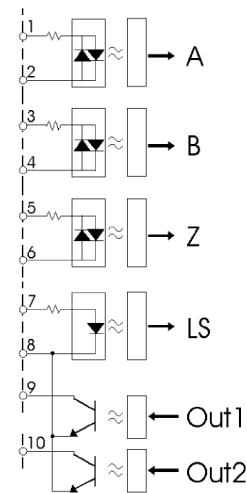
When you have finished configuring the module, click **OK** to save your changes and close the window.

A.10 OL2258 High Speed Pulse Counter

The OL2258 High Speed Counter Module provides for direct pulse counting for a variety of high speed pulse interface applications. Typical applications include motion control, metering and velocity measurement. The OL2258 contains on-board intelligence necessary for processing and counting pulse information as well as automatically triggering control outputs.

The OL2258 can be configured to operate in one of three pulse counting modes: Pulse & Direction, Up/Down Count, or Quadrature. Pulse & Direction and Up/Down Count will operate at up to 80KHz input pulse rates. Quadrature inputs count each quadrature state transitions at up to 160 KHz. Additionally, the OL2258 will return frequency information.

The counter has a 32-bit resolution and a total range of -2,147,483,648 to +2,147,483,648.

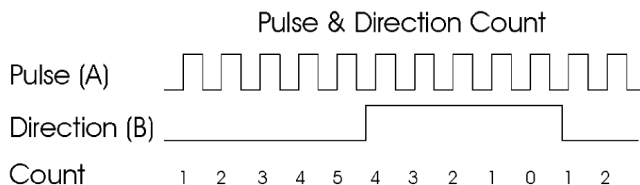


General Overview

The OL2258 is configurable. It can be used with pulse & direction, up/down count or quadrature type pulse encoders. These signals may come from shaft encoders, flow meters or any other signal source that produces a pulse train output. When operating, the OL2258 maintains a current cumulative count as a 32 bit integer value. It also makes available frequency snapshot data as the most recent count over either 1 second or 200 milliseconds. The Z and LS inputs can be used to automatically reset the count to a user defined value. Each transistor output can be configured to turn on when the count value is within its related count range.

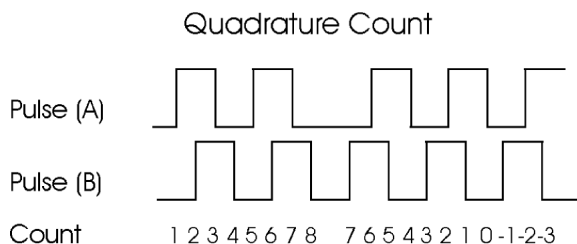
Pulse and Direction

In this configuration, pulses are input to "A". The counter direction is controlled by input "B". The operation is illustrated below.



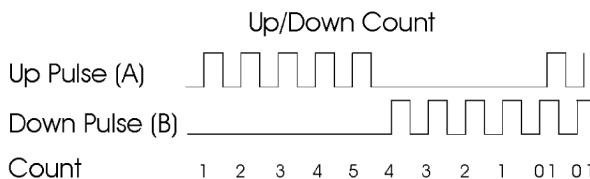
Quadrature Encoder Input

The counting process for quadrature type encoding is determined by the phase angle between input A and input B. If A leads B, the counter increments. If B leads A, the counter decrements. The count is incremented or decremented on each pulse transition as shown below.



Up/Down Count

For this type of configuration, the count increments on pulses input to "A" and decrements on pulses input to "B". This is illustrated in the figure below.



Z and LS Presetting

The count can be preset to a value that you define based on either or both inputs LS and Z. It can also be forced to a preset value on command via a message.

Through the configuration message, the counter can be set up to force a preset value when Z is active, LS is active, both Z and LS are active or on software command.

Output Control

The two open collector outputs can each be programmed to trigger within a programmable (via an ethernet message) count range. This range can be changed at any time via a "Send Output Range" message, effectively providing an unlimited number of ranges, under user program control.

Outputs will trigger within immediately, when the count enters the related range.

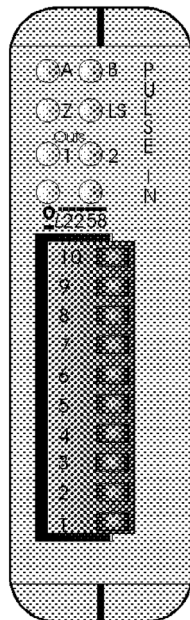
Frequency Measurement

Frequency data can be read back as a 16 bit signed integer value. The value will correspond to the most recent 1 second or 200 millisecond (configurable) pulse count.

Technical Specifications

Card Cage Power Required	400 mA
Inputs (all)	4
Pulse Inputs	2
Status Indicators	Logic Side LED
Counting Modes	Pulse & Direction, Up/Down Count, Quadrature
Count Value	32 Bit Signed
Frequency Data	16 Bit Signed (configurable for 1 sec or 200 msec)
Input Signal Type	Sinking, sourcing, or differential
Input Impedence	2.0K ohms nominal
Input frequency (on pulses)	80/160 KHz maximum
Min. Input On Voltage (or differential)	4.00V
Min. Input Off Voltage (or differential)	3.00V
Maximum Input Voltage	28V
Outputs	2
Output Type	Open collector
Commons	1 (connected internally)
Terminal Strip	Plug In (removable)
Terminal Screws	Slotted (0.1" blade max.)
Max. terminal wire gauge	18 AWG (use copper conductors)
Terminal block torque	2.2 lb-in
Required Temp. rating of field installed conductors	60°C / 75°C
Weight	1.24 oz (35g)
Type	82
Subtype	2

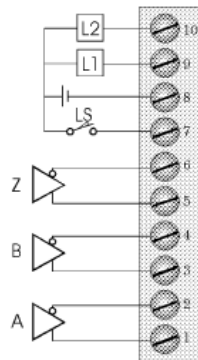
Connection Diagram



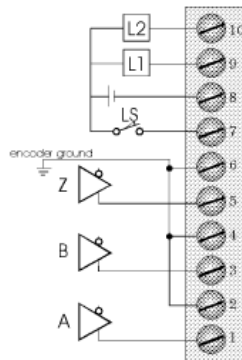
Terminals		
10	Out 2	Open collector output 2
9	Out 1	Open collector output 1
8	Common	Common for LS and two outputs
7	LS	Limit Switch input (optional)
6	Z2	Z input (optional)
5	Z1	
4	B2	Pulse input B (quadrature) / Direction input (pulse & direction) / Down pulse (up/down count)
3	B1	
2	A2	Pulse input A (quadrature) / Pulse input (pulse & direction) / Up pulse (up/down count)
1	A1	

The OL2258 High Speed Pulse counter is designed to interface to a variety of standard pulse encoder devices: differential, sourcing, or sinking. The figures below illustrate connections for each type of encoder.

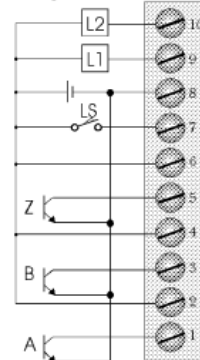
Differential Drive Interface
(Positive differential)



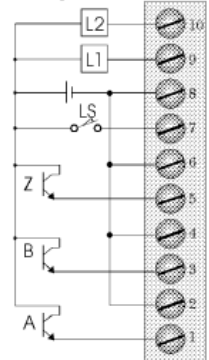
Differential Drive Interface
(Bipolar differential)



Sinking Encoder Interface



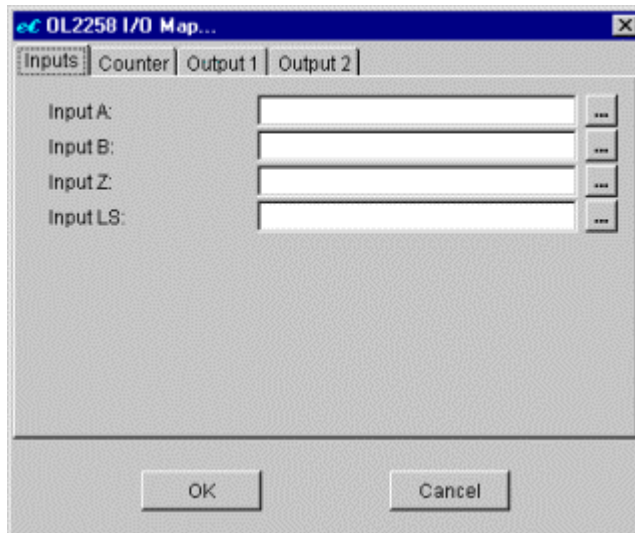
Sourcing Encoder Interface



A.10.1 OL2258 Configuration Options


The OL2258 module is configured through the Configure I/O menu command in the PointeControl development framework. (For more information on Configure I/O, see page 118.) Selecting an OL2258 module and clicking the **I/O** button opens the OL2258 I/O Map dialog window...

Inputs tab




Each of the four inputs on the OL2258 module is written to a separate bit.


- **Input A** – Bit to which the current status of Input A is written. When A becomes ON, the bit is set to 1. When A becomes OFF, the bit is set to 0. Mapped to an Input Bit tag (%IX).

To configure Input A, click the  to the right and select a tag from the pop-up menu. Each tag can be used only once.


- **Input B** – Bit to which the current status of Input B is written. When B becomes ON, the bit is set to 1. When B becomes OFF, the bit is set to 0. Mapped to an Input Bit tag (%IX).

To configure Input B, click the  to the right and select a tag from the pop-up menu. Each tag can be used only once.

- **Input Z** – Bit to which the current status of Input Z is written. When Z becomes ON, the bit is set to 1. When Z becomes OFF, the bit is set to 0. Mapped to an Input Bit tag (%IX).

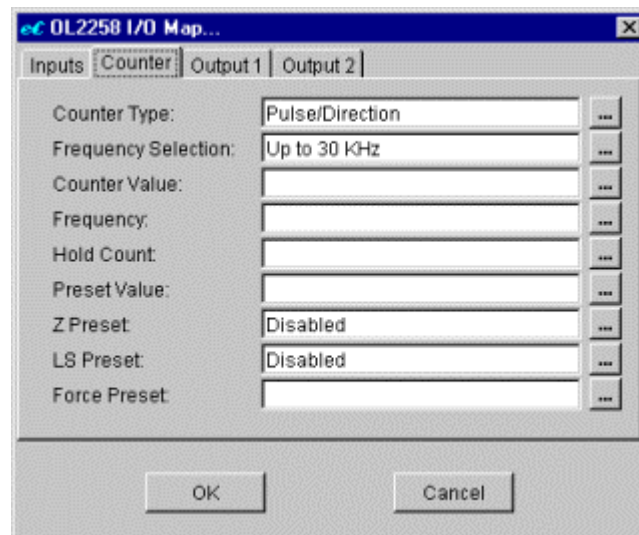
To configure Input Z, click the  to the right and select a tag from the pop-up menu. Each tag can be used only once.

- **Input LS** – Bit to which the current status of Input LS is written. When LS becomes ON, the bit is set to 1. When LS becomes OFF, the bit is set to 0. Mapped to an Input Bit tag (%IX).

To configure Input LS, click the  to the right and select a tag from the pop-up menu. Each tag can be used only once.


NOTE: These inputs do not necessarily need to be mapped to tags in Logic Memory for the pulse counter itself to function. The counter will still count pulses and write out the counter value, as configured in the Counter tab (below). However, it is sometimes useful to have access to these I/O points independent of the counter function.

Counter tab




General configuration parameters for pulse counter.

- **Counter Type** – The mode in which the counter will operate (see above). Options include:
 - Pulse/Direction (default) – In this mode, pulses are counted by Input A and the counter direction is determined by Input B.
 - Up/Down Count – In this mode, pulses received by Input A increment the counter and pulses received by Input B decrement the counter.
 - Quadrature – In this mode, counting is determined by the phase angle between Input A and Input B. If A leads B, then the counter increments. If B leads A, then the counter decrements.


To configure Counter Type, click the  to the right and select a mode from the pop-up menu.

- **Frequency Selection** – The frequency at which pulses will be received and counted. Options include:
 - Up to 30 KHz – The maximum pulse frequency will be less than or equal to 30 KHz.


- Over 30 KHz – The maximum pulse frequency will be greater than or equal to 30 KHz.

To configure Frequency Selection, click the  to the right and select a frequency option from the pop-up menu.


- **Counter Value** – Tag to which the current value of the pulse counter will be written. Mapped to a 32-bit Signed Input tag (%ID).

To configure Counter Value, click the  to the right and select a tag from the pop-up menu. Each tag can be used only once.


- **Frequency** – A sample of counted pulses taken over a predetermined time period. The length of the period depends on the Frequency Selection (above). If “Up to 30 KHz” is selected, then the period is the most recent 1 second. If “Over 30 KHz” is selected, then the period is the most recent 200 milliseconds (msecs). The total number of pulses counted during that period is written to the Frequency tag. Mapped to a 16-bit Signed Input tag (%IW).

To configure Frequency, click the  to the right and select a tag from the pop-up menu. Each tag can be used only once.


- **Hold Count** – Bit to pause and resume the counter. When the bit becomes ON, the counter is paused. When the bit becomes OFF, the counter is resumed from its last value. Mapped to an Output Bit tag (%QX).

To configure Hold Count, click the  to the right and select a tag from the pop-up menu. Each tag can be used only once.

- **Preset Value** – Value to which Counter Value will be forced whenever Z Preset, LS Preset, or Force Preset becomes ON (see below). Mapped to any 32-bit Signed tag (%ID, %MD, or %QD).


To configure Preset Value, click the  to the right and select a tag from the pop-up menu. Each tag can be used only once.

- **Z Preset** – Option to allow the status of Input Z to force the Preset Value.
 - If Disabled (default), then Input Z can be used as a generic DC input.
 - If Enabled, then Counter Value will be forced to Preset Value when Input Z becomes ON.


To configure Z Preset, click the  to the right and select an option (Disabled or Enabled) from the pop-up menu.

- **LS Preset** – Option to allow the status of Input LS to force the Preset Value.
 - If Disabled (default), then Input LS can be used as a generic DC input.

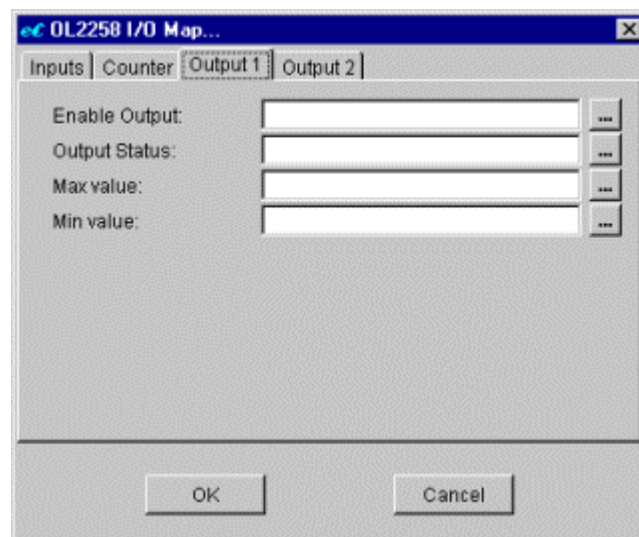
- If Enabled, then Counter Value will be forced to Preset Value when Input LS becomes ON.

To configure LS Preset, click the  to the right and select an option (Disabled or Enabled) from the pop-up menu.

- **Force Preset** – Bit to force Counter Value to Preset Value. Whenever the bit is set to 1, the value is forced. Mapped to an Output Bit tag (%QX).

To configure Force Preset, click the  to the right and select a tag from the pop-up menu. Each tag can be used only once.


Output tabs




The OL2258 module includes two (2) DC open collector outputs that are automatically turned on/off according on the current value of the pulse counter. When the counter value is inside an output’s specified range, the output is turned on. When the counter value goes outside (above or below) an output’s specified range, the output is turned off.

Each output is configured separately through its own tab: **Output 1** and **Output 2**. Both tabs have the same configuration parameters...


- **Enable Output** – Bit to enable and disable the output. When the bit is set to 1, the output is enabled. When the bit is set to 0, the output is disabled. Mapped to an Output Bit tag (%QX).

To configure Enable Output, click the  to the right and select a tag from the pop-up menu. Each tag can be used only once.


- **Output Status** – Bit to which the current status of the output is copied. (The actual status of the output is automatically controlled by the module itself.) When the output becomes ON, the bit is set to 1. When the output becomes OFF, the bit is set to 0. Mapped to an Input Bit tag (%IX).

To configure Output Status, click the  to the right and select a tag from the pop-up menu. Each tag can be used only once.

- **Max Value** – The maximum value of the range in which the output will be ON. When the counter value is above this maximum, the output will be OFF. Mapped to any 32-bit Signed tag (%IX, %MX, or %QX).

To configure Max Value, click the  to the right and select a tag from the pop-up menu. Each tag can be used only once.

- **Min Value** – The minimum value of the range in which the output will be ON. When the counter value is below this minimum, the output will be OFF. Mapped to any 32-bit Signed tag (%IX, %MX, or %QX).

To configure Min Value, click the  to the right and select a tag from the pop-up menu. Each tag can be used only once.

When you have finished configuring the module, click **OK** to save your changes and close the window.

A.11 OL2304 Analog Voltage Output Module

The OL2304 Analog Voltage Output Module provides four (4) output channels that are range configurable, on a channel by channel basis, to any of four common output ranges: 0-5 V, 0-10 V, +/-5 V, or +/-10 V. The voltage range for each channel can be configured through your PointeControl project rather than using physical jumpers. The module generates its own isolated output power supply, eliminating any need for an outside source.

To control the voltage output, the module receives an output value from the program and converts it to an actual voltage. The output value has a 12-bit resolution and is scaled from 0 to 4095 (0x0 to 0xFFF); i.e., the minimum voltage is equal to 0, the maximum voltage is equal to 4095, and all other voltages are scaled in between.

To find the correct output value for a given voltage, you must configure a Flow Chart block or Ladder Diagram rung to perform the following calculations:

- For a range of 0-5 V...

$$\text{Output Value} = (\text{Actual Voltage} \times 4095) / 5$$

- For a range of 0-10 V...

$$\text{Output Value} = (\text{Actual Voltage} \times 4095) / 10$$

- For a range of +/-5 V...

$$\text{Output Value} = [(\text{Actual Voltage} + 5) / 10] \times 4095$$

- For a range of +/-10 V...

$$\text{Output Value} = [(\text{Actual Voltage} + 10) / 20] \times 4095$$

Example 1: A channel is configured for a range of 0-10 V and a 3.3V output is required. The output value is calculated as...

$$\text{Output Value} = (3.3 \times 4095) / 10 = \mathbf{1351}$$

Example 2: A channel is configured for a range of +/-5 V and a -2.5V output is required. The output value is calculated as...

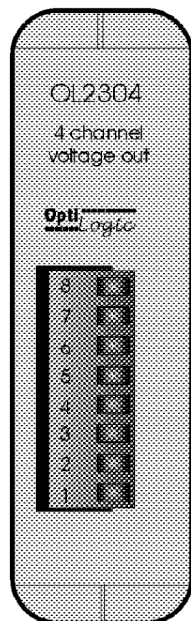
$$\text{Output Value} = [(-2.5 + 5) / 10] \times 4095 = \mathbf{1024}$$

Technical Specifications

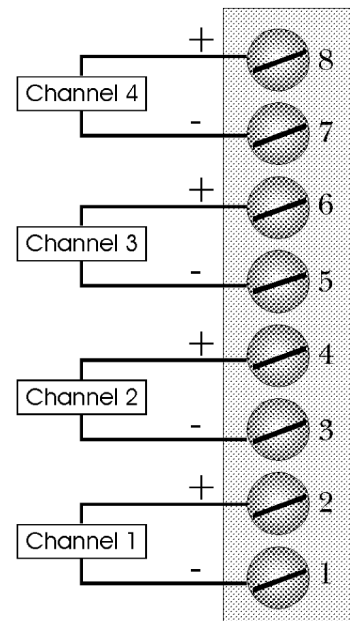
Card Cage Power Required	700 mA
Ouptuts	4
Output Ranges	0-5V, 0-10V, +/-5V, +/-10V (configurable by channel)
Resolution	12 bit (1 in 4096)
Output Type	Single-ended, 1 common

External Power Required	none
Output Current	+/- 5 mA
Short Circuit Current	+/- 15 mA
Offset Calibration Error	+/- 32 counts @ 0-5V +/- 16 counts @ 0-10V +/- 16 counts @ +/-5V +/- 8 counts @ +/-10V
Nonlinearity	+/- 1 count
Terminal Strip	Plug In (removable)
Terminal Screws	Slotted (0.1" blade max.)
Max. terminal wire gauge	18 AWG (use copper conductors)
Terminal block torque	2.2 lb-in
Required Temp. rating of field installed conductors	60°C / 75°C
Type	25
Subtype	1

Connection Diagram

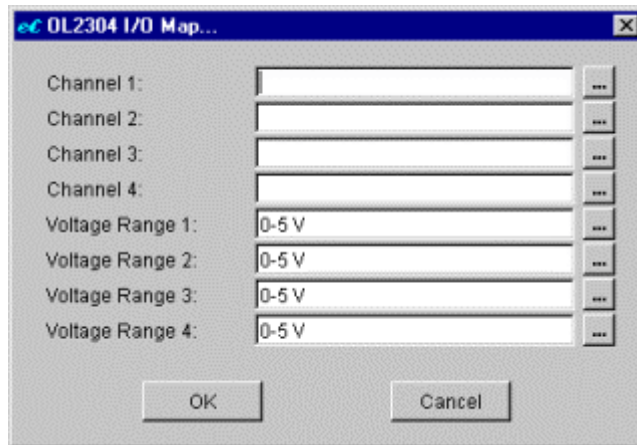


Terminal	
8	Out 4
7	Common
6	Out 3
5	Common
4	Out 2
3	Common
2	Out 1
1	Common



A.11.1 OL2304 Configuration Options

The OL2304 module is configured through the Configure I/O menu command in the PointeControl development framework. (For more information on Configure I/O, see page 118.) Selecting an OL2304 module and clicking the **I/O** button opens the OL2304 I/O Map dialog window...



Each of the four output channels on the OL2304 module can be configured separately. To use a channel, you must configure both its **Channel** and its **Voltage Range**.

- **Channel *n*** – Tag from which the output value for the specified channel is taken. Mapped to a 16-bit Unsigned Output tag (%QUW).

To configure a Channel, click the to the right and select a tag from the pop-up menu. Each tag can be used only once.

- **Voltage Range *n*** – Voltage range for the specified channel. This option is configured separately for each output channel. Options include 0-5 V (default), 0-10 V, +/-5 V or +/-10 V.

To configure a Voltage Range, click the to the right and select a range from the pop-up menu.

NOTE: You can leave any or all output channels unconfigured, but if you do, those I/O points will not be available to your project.

When you have finished configuring the module, click **OK** to save your changes and close the window.

A.12 OL2408 Analog Voltage Input

The OL2408 Analog Voltage Input Module provides eight (8) voltage sensors. Each sensor reads the current input voltage, scales it, and writes the scaled value to a Logic Memory tag. The value has 14-bit resolution and is scaled from 0 to 16383 (0x0 to 0x3FFF); i.e., the minimum voltage is equal to 0, the maximum voltage is equal to 16383, and all other voltages are scaled in between.

NOTE: The OL2408 comes factory configured for 0-5VDC input range and cannot be changed through software. If you need 0-10VDC input range, you must physically set a jumper on the module board.

To convert the scaled value back to an actual voltage input, you must configure a Flow Chart block or Ladder Logic rung to perform the following calculation:

$$\text{Voltage Input} = \text{Maximum Voltage} \times (\text{Scaled Value} / 16383)$$

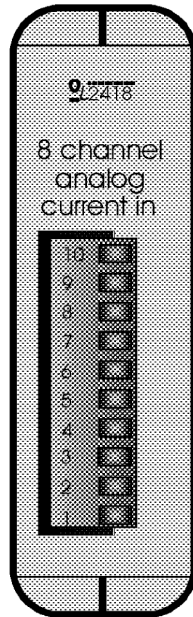
Remember that Maximum Voltage can be either 5 or 10, depending on the jumper setting.

Technical Specifications

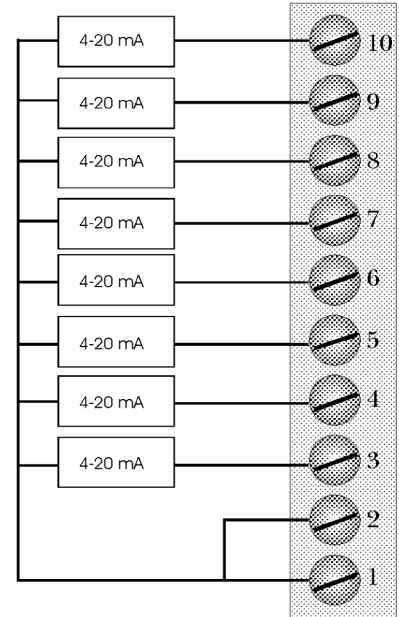
Card Cage Power Required	700 mA
Inputs	4
Input Type	0-5 VDC or 0-10 VDC
Input Impedence	10 MOhm
Maximum Voltage Input	+/- 15VDC
Conversion Type	Successive approximation
Resolution	14 bit (1 in 16384)
Full Scale Calibration Error	+/- 15 counts maximum +/- 5 counts typical
Offset Calibration Error	+/- 2 counts maximum
Linearity Error	+/- 1.25 count maximum
Input Stability	+/- 2 counts
Terminal Strip	Plug In (removable)
Terminal Screws	Slotted (0.1" blade max.)
Max. terminal wire gauge	18 AWG (use copper conductors)
Terminal block torque	2.2 lb-in
Required Temp. rating of field installed conductors	60°C / 75°C
Type	17

Subtype	1
----------------	---

Connection Diagram

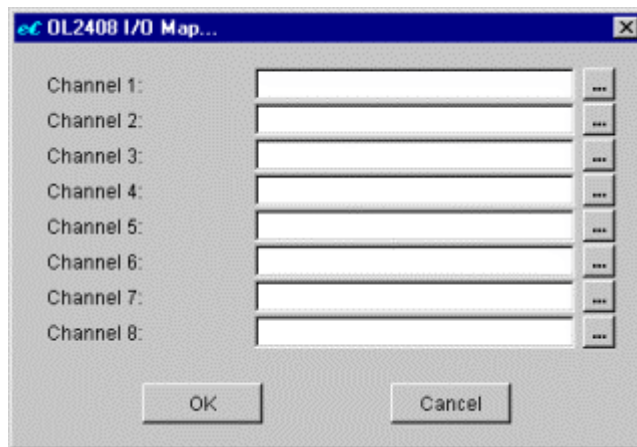


Terminal	
10	Channel 8
9	Channel 7
8	Channel 6
7	Channel 5
6	Channel 4
5	Channel 3
4	Channel 2
3	Channel 1
2	Common
1	Common




A.12.1 OL2408 Configuration Options

The OL2408 module is configured through the Configure I/O menu command in the PointeControl development framework. (For more information on Configure I/O, see page 118.) Selecting an OL2408 module and clicking the **I/O** button opens the OL2408 I/O Map dialog window...



The OL2408 module has eight input channels. Each channel is configured separately.

Channel n – Tag to which the scaled value of the input voltage is written. Mapped to a 16-bit Unsigned Input tag (%IUW).

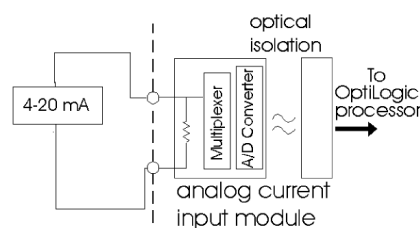
To configure a Channel, click the  to the right and select a tag from the pop-up menu. Each tag can be used only once.

NOTE: You can leave any or all Channels unconfigured, but if you do, those I/O points will not be available to your project.

When you have finished configuring the module, click **OK** to save your changes and close the window.

A.13 OL2418 Analog Current Input

The OL2418 Analog Current Input Module provides eight (8) amperage sensors. Each sensor reads the current input amperage, scales it, and writes the scaled value to a Logic Memory tag. The value has 14-bit resolution and is scaled from 0 to 16383 (0x0 to 0x3FFF); i.e., the minimum amperage is equal to 0, the maximum amperage is equal to 16383, and all other amperages are scaled in between.



NOTE: The OL2408 module is currently designed for an amperage range of 4 to 20 mA. This range cannot be changed.

To convert the scaled value back to an actual amperage input, you must configure a Flow Chart block or Ladder Logic rung to perform the following calculation:

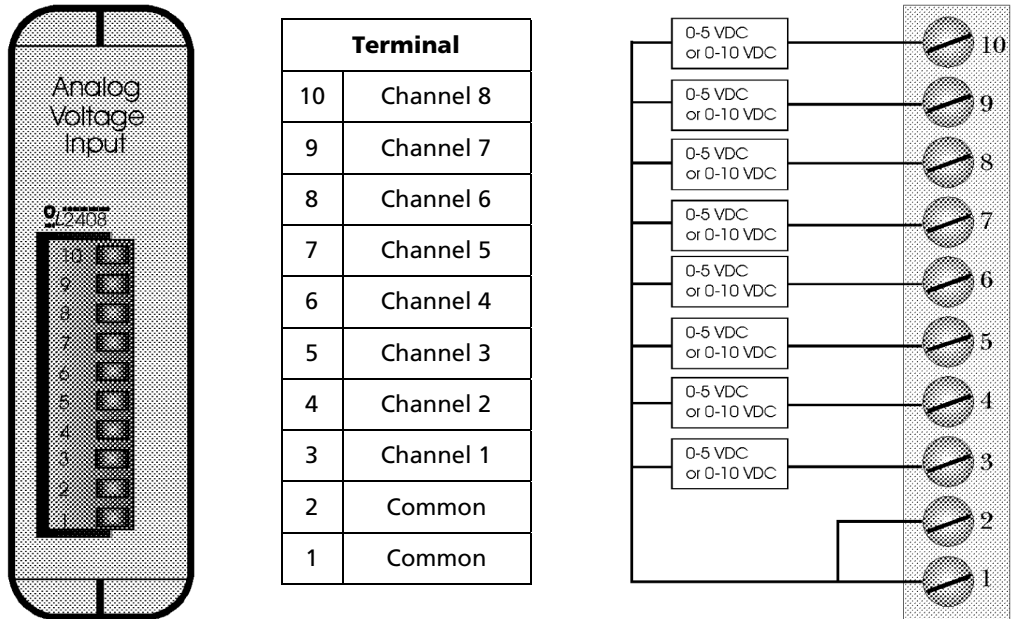
$$\text{Amperage Input} = [16 \times (\text{Scaled Value} / 16383)] + 4$$

Technical Specifications

Card Cage Power Required	700 mA
Inputs	8
Input Type	4-20 mA
Input Impedence	250 Ohms +/- 0.05%
Conversion Type	Successive approximation
Resolution	14 bit (1 in 16384)
Full Scale Calibration Error	+/- 15 counts maximum +/- 5 counts typical
Offset Calibration Error	+/- 2 counts maximum
Power Isolation	Transformer
Signal Isolation	Optical
Terminal Strip	Plug In (removable)
Terminal Screws	Slotted (0.1" blade max.)
Max. terminal wire gauge	18 AWG (use copper conductors)
Terminal block torque	2.2 lb-in
Required Temp. rating of field installed conductors	60°C / 75°C
Type	18

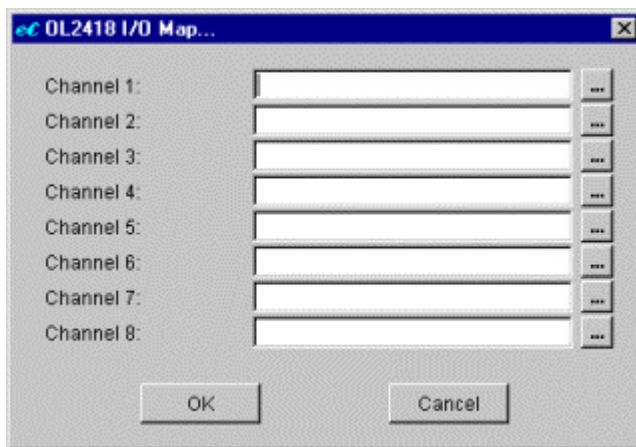
Subtype	2
---------	---

Connection Diagram




A.13.1 OL2418 Configuration Options

The OL2418 module is configured through the Configure I/O menu command in the PointeControl development framework. (For more information on Configure I/O, see page 118.) Selecting an OL2418 module and clicking the **I/O** button opens the OL2418 I/O Map dialog window...



The OL2418 module has eight input channels. Each channel is configured separately.

Channel n – Tag to which the scaled value of the input amperage is written. Mapped to a 16-bit Unsigned Input tag (%IUW).

To configure a Channel, click the  to the right and select a tag from the pop-up menu. Each tag can be used only once.

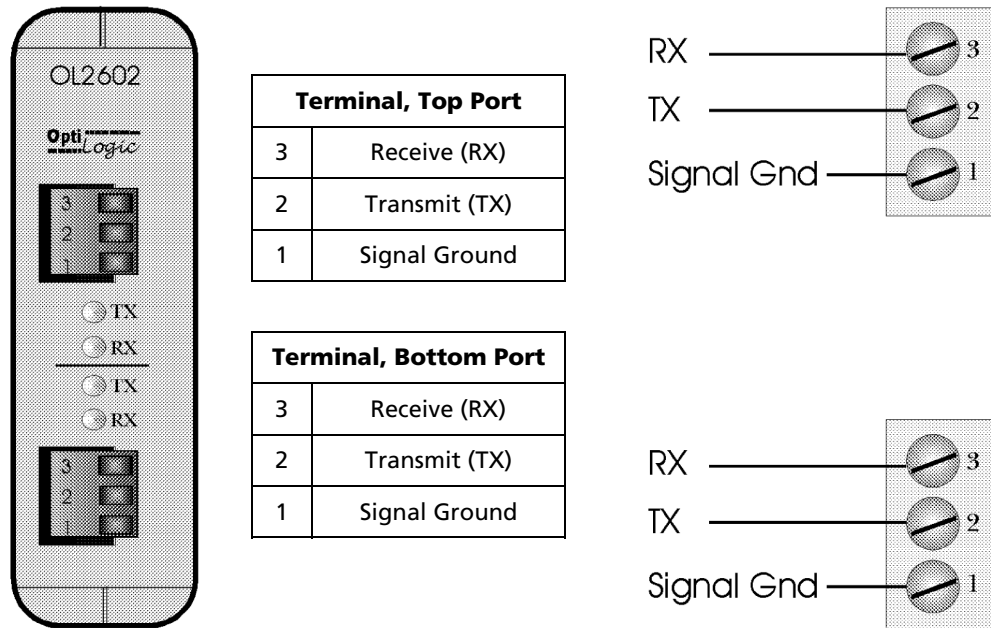
NOTE: You can leave any or all Channels unconfigured, but if you do, those I/O points will not be available to your project.

When you have finished configuring the module, click **OK** to save your changes and close the window.

A.14 OL2602 Dual Serial Port Module

The OL2602 Dual Serial Port Module provides two (2) standard RS232 serial ports – in addition to the built-in port on the Pointe Controller unit itself – that can be used to connect to and communicate with a wide variety of device networks and control hardware.

Connection Diagram



Technical Specifications

Card Cage Power Required	110 mA
Communications Ports	2
Type	RS232C
Baud Rates	1200, 2400, 4800, 9600, 19200 (selectable)
Parity	Even, odd or none
Data Bits	7 or 8
Transmit Buffer	48 bytes
Receive Buffer	48 bytes
Terminal Strip	Plug In (removable)
Terminal Screws	Slotted (0.1" blade max.)
Max. terminal wire gauge	18 AWG (use copper conductors)

Terminal block torque	2.2 lb-in
Required Temp. rating of field installed conductors	60°C / 75°C
Weight	1.0 oz (29g)
Type	112
Subtype	2

A.14.1 OL2602 Configuration Options

This module cannot be configured through the Configure I/O dialog. To configure and use this module, see [Serial Commands](#) on page 321.

A.15 OL3406 Pushbutton/Indicator Panel

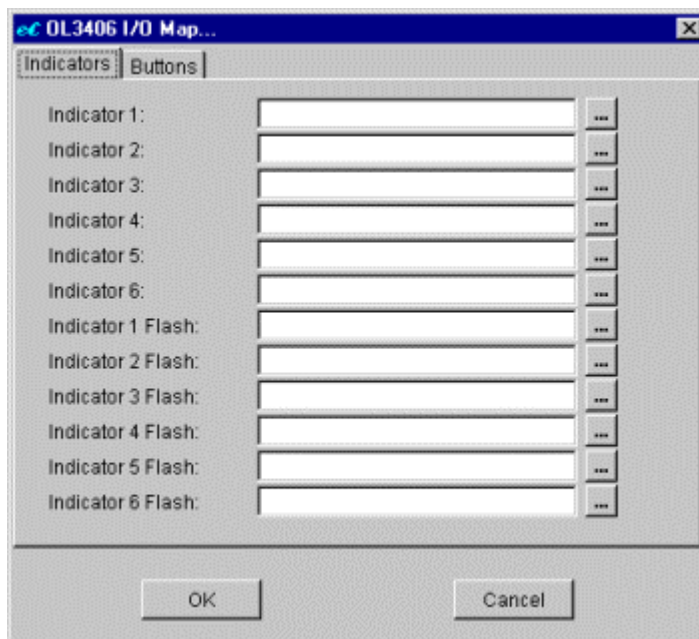
The OL3406 Pushbutton/Indicator Panel has four user-definable pushbuttons and six white indicator bars. The buttons can be configured for either momentary or alternate-action operation. The button LEDs normally reflect button on/off status. The momentary buttons can also be configured for LED separation (direct on/off control). Every button LED and indicator bar can be turned on, off, or flashed.



A.15.1 OL3406 Configuration Options


The OL3406 panel is configured through the Configure I/O menu command in the PointeControl development framework. (For more information on Configure I/O, see page 118.) Selecting an OL3406 panel and clicking the **I/O** button opens the OL3406 I/O Map dialog window...

Indicators tab




This tab configures the six white indicator bars on the OL3406 panel.

- **Indicator *n*** – Bit that turns the indicator on/off. When the bit becomes ON, the indicator is turned on. When the bit becomes OFF, the indicator is turned OFF. Mapped to an Output Bit tag (%QX).

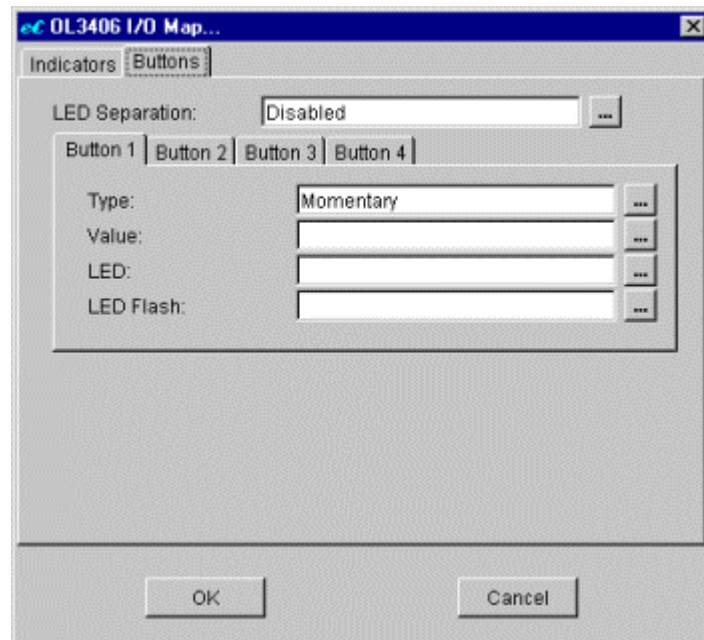
To configure an Indicator, click the  to the right and select a tag from the pop-up menu. Each tag can be used only once.

- **Indicator *n* Flash** – Bit that quickly flashes the indicator. (The indicator itself must already be on.) When the bit becomes ON, the indicator starts flashing. When the bit becomes OFF, the indicator stops flashing. Mapped to an Output Bit (%QX).

To configure an Indicator Flash, click the  to the right and select a tag from the pop-up menu. Each tag can be used only once.

NOTE: You can leave any or all Indicators unconfigured, but if you do, those I/O points will not be available to your project.


Buttons tab



This tab controls the four white pushbuttons on the OL3406 panel.

- **LED Separation** – Option to control the red LEDs embedded in the pushbuttons, separate from the actual ON/OFF states of the buttons. (The setting applies to all four buttons.)
 - If Disabled (default), then each button’s LED directly reflects the current ON/OFF state of the button.


- If Enabled, then each button LED can be turned on/off independently using the LED parameter (below).

To configure LED Separation, click the  to the right and select an option (Disabled or Enabled) from the pop-up menu.


NOTE: This configuration applies to a button only if the button is also set Momentary (see below).

Button n – Each pushbutton can be configured separately, using its own Button sub-tab. The buttons on the panel are numbered 1 through 4, from left to right. All four sub-tabs have the same parameters:

- **Type** – Option to change the responsiveness of the button.
 - If Momentary (default), then the button is ON only so long as it is pressed and held by the operator.
 - If Alternate Action, then the button toggles between ON and OFF every time it is pressed by the operator.


To configure Type, click the  to the right and select an option (Momentary or Alternate Action) from the pop-up menu.

- **Value** – The current ON/OFF state of the button. Mapped to an Input Bit tag (%IX).


To configure Value, click the  to the right and select a tag from the pop-up menu. Each tag can be used only once.

NOTE: The current value of the button should always be read as an *input*; the associated tag should never be directly set by the program logic. To change the state of the button from within the program, use the **Button On and Button Off commands**.

- **LED** – Bit that turns the button LED on/off, if LED Separation is enabled and the button is set Momentary (see above). Mapped to an Output Bit tag (%QX).

To configure LED, click the  to the right and select a tag from the pop-up menu. Each tag can be used only once.

- **LED Flash** – Bit that quickly flashes the button LED, if LED Separation is enabled and the button is set Momentary (see above). The LED must already be on before it can be flashed. Mapped to an Output Bit tag (%QX).

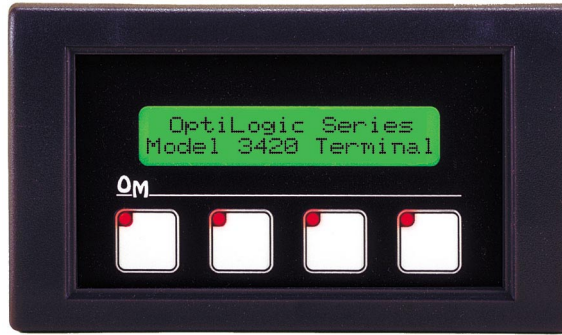
To configure LED Flash, click the  to the right and select a tag from the pop-up menu. Each tag can be used only once.

NOTE: You can leave any or all Buttons unconfigured, but if you do, those I/O points will not be available to your project.

When you have finished configuring the panel, click **OK** to save your changes and close the window.

A.16 OL3420 Operator Terminal

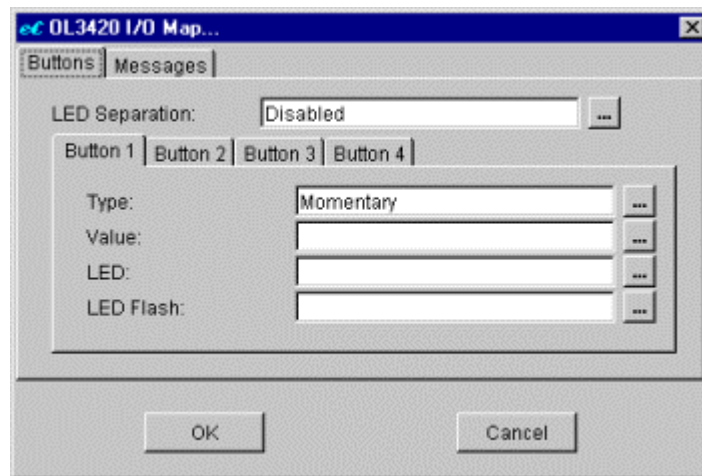
The OL3420 Operator Terminal has four user-definable pushbuttons and a 2 line x 20 character LCD display. The buttons can be configured for either momentary or alternate-action operation. The button LEDs normally reflect button on/off status. The momentary buttons can also be configured for LED separation (direct on/off control). Every button LED can be turned on, off, or flashed.



A.16.1 OL3420 Configuration Options

The OL3420 panel is configured through the Configure I/O menu command in the PointeControl development framework. (For more information on Configure I/O, see page 118.) Selecting an OL3420 panel and clicking the **I/O** button opens the OL3420 I/O Map dialog window...


Buttons tab



This tab controls the four white pushbuttons on the OL3420 panel.

- **LED Separation** – Option to control the red LEDs embedded in the pushbuttons, separate from the actual ON/OFF states of the buttons. (The setting applies to all four buttons.)


- If Disabled (default), then each button's LED directly reflects the current ON/OFF state of the button.
- If Enabled, then each button LED can be turned on/off independently using the LED parameter (below).

To configure LED Separation, click the  to the right and select an option (Disabled or Enabled) from the pop-up menu.


NOTE: This configuration applies to a button only if the button is also set Momentary (see below).

Button *n* – Each pushbutton can be configured separately, using its own Button sub-tab. The buttons on the panel are numbered 1 through 4, from left to right. All four sub-tabs have the same parameters:

- **Type** – Option to change the responsiveness of the button.
 - If Momentary (default), then the button is ON only so long as it is pressed and held by the operator.
 - If Alternate Action, then the button toggles between ON and OFF every time it is pressed by the operator.


To configure Type, click the  to the right and select an option (Momentary or Alternate Action) from the pop-up menu.

- **Value** – The current ON/OFF state of the button. Mapped to an Input Bit tag (%IX).


To configure Value, click the  to the right and select a tag from the pop-up menu. Each tag can be used only once.

NOTE: The current value of the button should always be read as an *input*; the associated tag should never be directly set by the program logic. To change the state of the button from within the program, use the [Button On and Button Off commands](#).

- **LED** – Bit that turns the button LED on/off, if LED Separation is enabled and the button is set Momentary (see above). Mapped to an Output Bit tag (%QX).

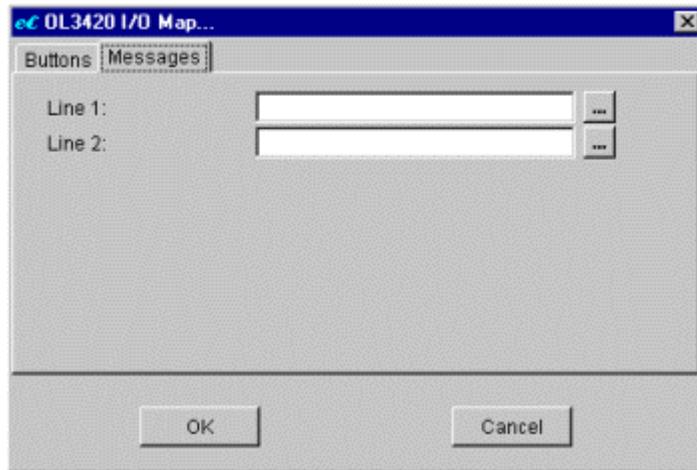
To configure LED, click the  to the right and select a tag from the pop-up menu. Each tag can be used only once.

- **LED Flash** – Bit that quickly flashes the button LED, if LED Separation is enabled and the button is set Momentary (see above). The LED must already be on before it can be flashed. Mapped to an Output Bit tag (%QX).


To configure LED Flash, click the  to the right and select a tag from the pop-up menu. Each tag can be used only once.

NOTE: You can leave any or all Buttons unconfigured, but if you do, those I/O points will not be available to your project.

Messages tab



This tab controls two lines of the LCD on the OL3420 panel. The lines on the panel are numbered 1 and 2, from top to bottom. Each line is mapped to a separate 20-character String variable.

To configure a **Line**, click the  to the right and select a String variable from the pop-up menu. Each variable can be used only once.

Lines can be set and changed independently of each other; text does not wrap from one line to the next. To display text on the panel, use String commands ([Flow Chart](#) or [Ladder Block](#)) to write the desired text to the associated String variables.

NOTE: You can leave any or all Lines unconfigured, but if you do, those I/O points will not be available to your project.

When you have finished configuring the panel, click **OK** to save your changes and close the window.

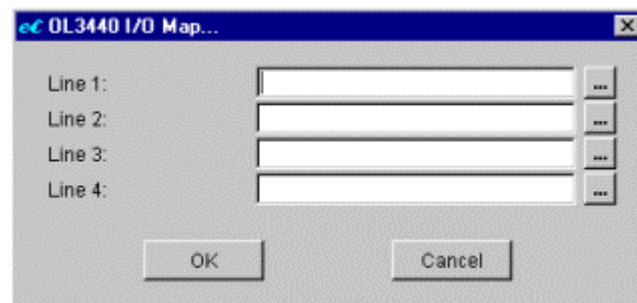
A.17 OL3440 Display Panel

The OL3440 Display Panel is a 4 line x 20 character LCD display. You can send text to any line.



A.17.1 OL3440 Configuration Options

The OL3440 panel is configured through the Configure I/O menu command in the PointeControl development framework. (For more information on Configure I/O, see page 118.) Selecting an OL3440 panel and clicking the **I/O** button opens the OL3440 I/O Map dialog window...



This tab controls to four lines of the LCD on the OL3440 panel. The lines on the panel are numbered 1 through 4, from top to bottom. Each line is mapped to a separate 20-character String variable.

To configure a **Line**, click the **...** to the right and select a String variable from the pop-up menu. Each variable can be used only once.

Lines can be set and changed independently of each other; text does not wrap from one line to the next. To display text on the panel, use String commands ([Flow Chart](#) or [Ladder Block](#)) to write the desired text to the associated String variables.

NOTE: You can leave any or all Lines unconfigured, but if you do, those I/O points will not be available to your project.

When you have finished configuring the panel, click **OK** to save your changes and close the window.

A.18 OL3850 Keypad Terminal

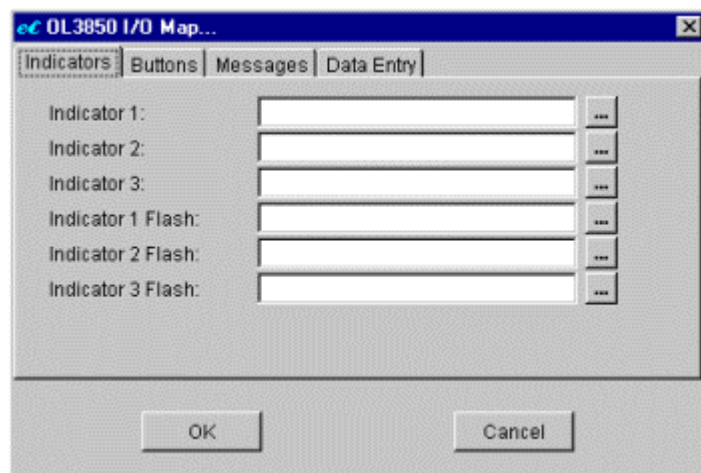
The OL3850 Keypad Terminal has a 2 line x 20 character LCD display, a numeric keypad w/ up and down arrows, five user-definable pushbuttons, and three colored indicator bars. The buttons can be configured for either momentary or alternate-action operation. The button LEDs normally reflect button on/off status. The momentary buttons can also be configured for LED separation (direct on/off control). Every button LED and indicator bar can be turned on, off, or flashed.



A.18.1 OL3850 Configuration Options


The OL3850 panel is configured through the Configure I/O menu command in the PointeControl development framework. (For more information on Configure I/O, see page 118.) Selecting an OL3850 panel and clicking the **I/O** button opens the OL3850 I/O Map dialog window...

Indicators tab




This tab configures the green (Indicator 1), yellow (Indicator 2), and red (Indicator 3) bars on the OL3850 panel.

- **Indicator *n*** – Bit that turns the indicator on/off. When the bit becomes ON, the indicator is turned on. When the bit becomes OFF, the indicator is turned OFF. Mapped to an Output Bit tag (%QX).

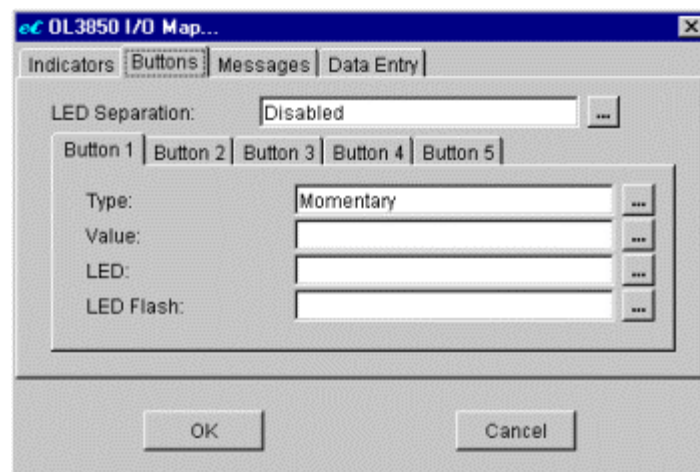
To configure an Indicator, click the  to the right and select a tag from the pop-up menu. Each tag can be used only once.

- **Indicator *n* Flash** – Bit that quickly flashes the indicator. (The indicator itself must already be on.) When the bit becomes ON, the indicator starts flashing. When the bit becomes OFF, the indicator stops flashing. Mapped to an Output Bit (%QX).

To configure an Indicator Flash, click the  to the right and select a tag from the pop-up menu. Each tag can be used only once.


NOTE: You can leave any or all Indicators unconfigured, but if you do, those I/O points will not be available to your project.

Buttons tab



This tab controls the five white pushbuttons on the OL3850 panel.


- **LED Separation** – Option to control the red LEDs embedded in the pushbuttons, separate from the actual ON/OFF states of the buttons. (The setting applies to all five buttons.)
 - If Disabled (default), then each button’s LED directly reflects the current ON/OFF state of the button.
 - If Enabled, then each button LED can be turned on/off independently using the LED parameter (below).

To configure LED Separation, click the  to the right and select an option (Disabled or Enabled) from the pop-up menu.


NOTE: This configuration applies to a button only if the button is also set Momentary (see below).

Button n – Each pushbutton can be configured separately, using its own Button sub-tab. The buttons on the panel are numbered 1 through 5, from left to right. All five sub-tabs have the same parameters:

- **Type** – Option to change the responsiveness of the button.
 - If Momentary (default), then the button is ON only so long as it is pressed and held by the operator.
 - If Alternate Action, then the button toggles between ON and OFF every time it is pressed by the operator.


To configure Type, click the  to the right and select an option (Momentary or Alternate Action) from the pop-up menu.

- **Value** – The current ON/OFF state of the button. Mapped to an Input Bit tag (%IX).

To configure Value, click the  to the right and select a tag from the pop-up menu. Each tag can be used only once.

NOTE: The current value of the button should always be read as an *input*; the associated tag should never be directly set by the program logic. To change the state of the button from within the program, use the [Button On and Button Off commands](#).

- **LED** – Bit that turns the button LED on/off, if LED Separation is enabled and the button is set Momentary (see above). Mapped to an Output Bit tag (%QX).

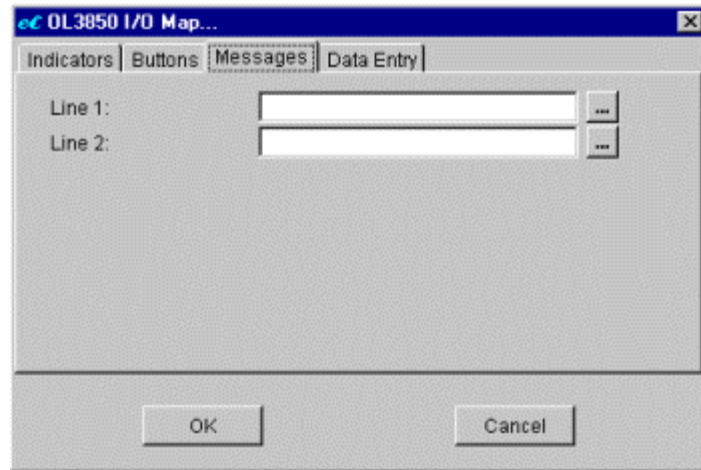
To configure LED, click the  to the right and select a tag from the pop-up menu. Each tag can be used only once.

- **LED Flash** – Bit that quickly flashes the button LED, if LED Separation is enabled and the button is set Momentary (see above). The LED must already be on before it can be flashed. Mapped to an Output Bit tag (%QX).

To configure LED Flash, click the  to the right and select a tag from the pop-up menu. Each tag can be used only once.

NOTE: You can leave any or all Buttons unconfigured, but if you do, those I/O points will not be available to your project.

Messages tab



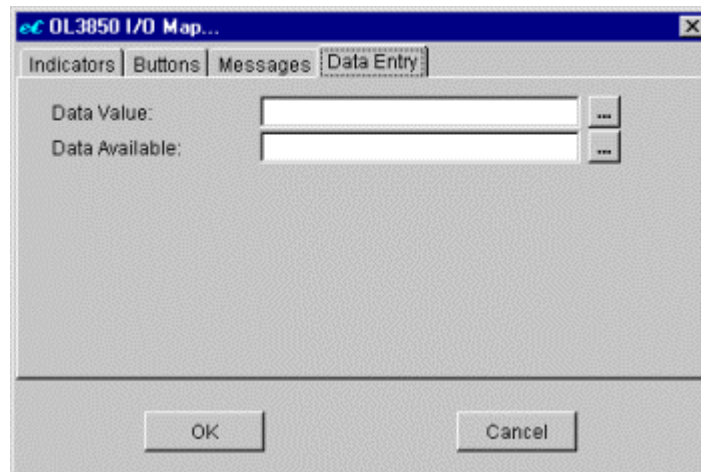
This tab controls to two lines of the LCD on the OL3850 panel. The lines on the panel are numbered 1 and 2, from top to bottom. Each line is mapped to a separate 20-character String variable.

To configure a **Line**, click the **...** to the right and select a String variable from the pop-up menu. Each variable can be used only once.

Lines can be set and changed independently of each other; text does not wrap from one line to the next. To display text on the panel, use String commands ([Flow Chart](#) or [Ladder Block](#)) to write the desired text to the associated String variables.


NOTE: You can leave any or all Lines unconfigured, but if you do, those I/O points will not be available to your project.

Data Entry tab




This tab is used to configure two variables that are required to enter data through the numeric keypad on the OL3850 panel.

- **Data Value** – When the user inputs a numeric value and presses the ENTER key on the keypad, the value is saved to Data Value. Mapped to a 32-bit Real Input tag (%IF).

To configure Data Value, click the  to the right and select a tag from the pop-up menu. Each tag can be used only once.

- **Data Available** – When the user inputs a numeric value and presses the ENTER key on the keypad, the Data Available flag is set. Mapped to an Input Bit tag (%IX).

To configure Data Available, click the  to the right and select a tag from the pop-up menu. Each tag can be used only once.

For more information on entering data through the numeric keypad, see [Operator Panel Commands](#) on page 333.

When you have finished configuring the panel, click **OK** to save your changes and close the window.

Appendix B: Flow Chart Command Reference

This appendix provides complete descriptions and configuration instructions for all of the Flow Chart process commands that are available in the PointeControl development framework. (For more information on building Flow Charts, see page 129.)

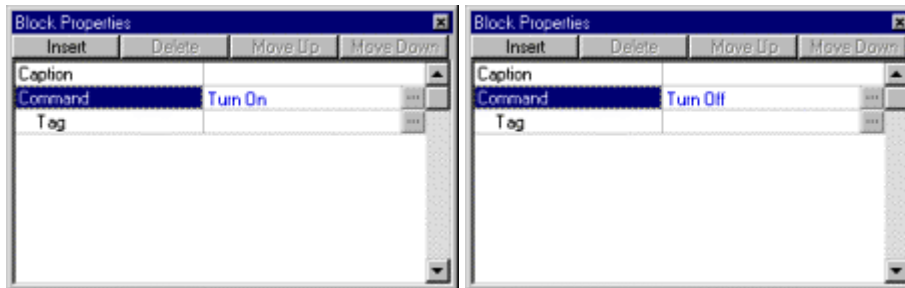
TIP: The information provided in this appendix is also available via the PointeControl Framework online help. To access the help, choose **Contents** from the Framework's **Help** menu.

B.1 General Commands

General commands (in a **Process block**) are used to set and clear Logic Memory tags, to enable and disable Flow Charts, and to insert a timed delay in the chart execution.

B.1.1 Turn On and Turn Off

These commands can be selected from the **General commands** list.



When used in a Flow Chart, the Turn On and Turn Off commands set a specified Bit tag to 1 or 0, respectively.

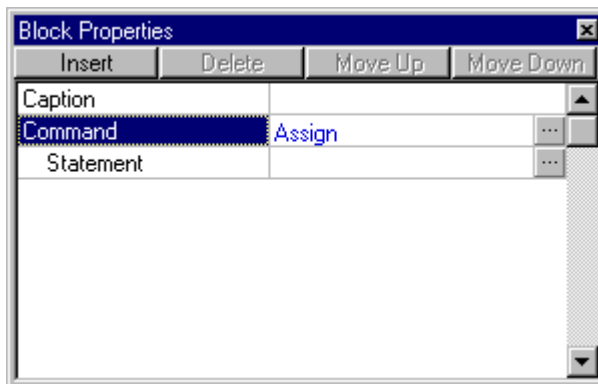
Parameters for these commands include:

- **Tag** – The Bit tag to be turned on or off by the command. Bit-type tags include Input Bits (%IX), Memory Bits (%MX), and Output Bits (%QX).

To configure the Tag parameter, click the **...** button to open a standard **Select Tag** dialog.

B.1.2 Assign

This command can be selected from the **General commands** list.




When used in a Flow Chart, the Assign command writes a value to a specified tag.

Parameters for this command include:

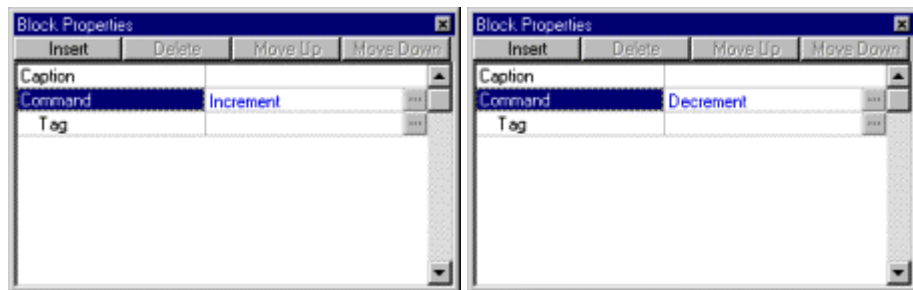
- **Statement** – The statement which specifies the tag to be set and describes the value to be written.

The value of may be an Input/Memory/Output tag, a literal numeric value, or some other logical expression that is evaluated every time the block is executed.

To configure the Statement parameter, click the  button to open a standard [Build Assignment](#) dialog.

B.1.3 Increment and Decrement


These commands can be selected from the [General commands](#) list.



When used in a Flow Chart, the Increment and Decrement commands increase or decrease a specified non-Bit tag's value by 1.

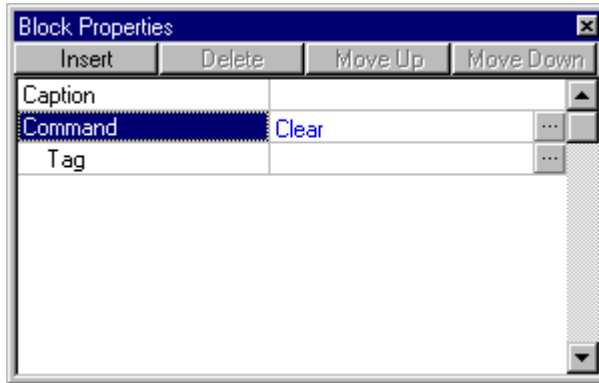
Parameters for this command include:

- **Tag** – The tag to be incremented or decremented by the command. Any non-Bit Input, Memory, or Output tag can be specified. String and Timer variables, as well as Bit tags, cannot be specified.

To configure the Tag parameter, click the  button to open a standard [Select Tag](#) dialog.

B.1.4 Clear


This command can be selected from the **General commands** list.



When used in a Flow Chart, the Clear command resets a specified tag to 0.

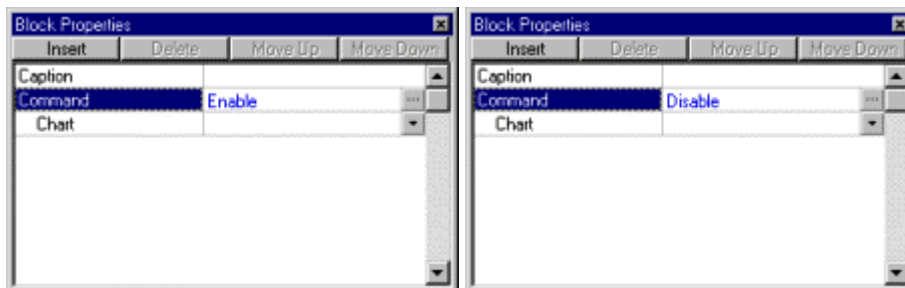
Parameters for this command include:

- **Tag** – The tag to be reset to 0 by the command. Any Input, Memory, or Output tag can be specified. Strings and Timers cannot be specified.

To configure the Tag parameter, click the  button to open a standard **Select Tag** dialog.

B.1.5 Enable and Disable


These commands can be selected from the **General commands** list.



When used in a Flow Chart, the Enable and Disable commands change whether a specified Flow Chart is scanned. An enabled chart is scanned normally, while a disabled chart is not scanned. Using these commands is equivalent to toggling the Default State property in the specified chart's **Start block**.

Parameters for this command include:

- **Chart** – The Flow Chart to be enabled or disabled by the command. Any regular Flow Chart can be enabled or disabled. Ladder Diagrams, as well as Flow Charts which have been made Subcharts, cannot be enabled or disabled.

To configure the Chart parameter, click the  button and select a chart from the drop-down menu.

B.1.6 Get Tag Name


This command can be selected from the [General commands](#) list.




When used in a Flow Chart, the Get Tag Name command retrieves the name (alias) of a specified tag and saves it to a string.

Parameters for this command include:

- **Source Tag** – The tag from which the name is retrieved. Any Input, Memory, or Output tag can be selected, as well as any String variable. Timers cannot be selected.

To configure the Source Tag parameter, click the  button to open a standard [Select Tag](#) dialog.

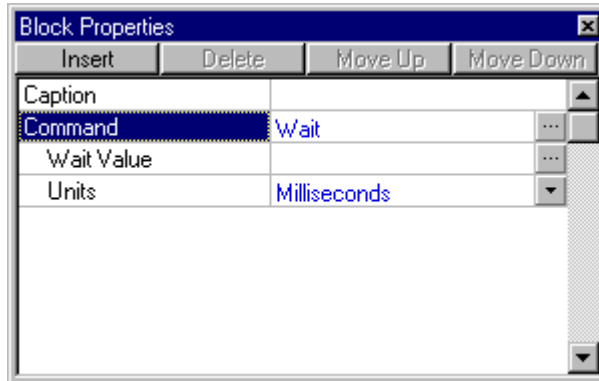
- **Destination String** – The String tag to which the name of Source Tag is written. If the name is longer than the defined Element Length of the String, then it will be truncated.

To configure the Destination String parameter, click the  button to open a standard [Select String Tag](#) dialog.

- **Name Type** – Since all tags are referenced by Alias, this parameter cannot be configured in PointeControl.

B.1.7 Wait

This command can be selected from the [General commands](#) list.




When used in a Flow Chart, the Wait command inserts a delay of specified duration into the execution of the chart.


Parameters for this command include:

- **Wait Value** – The number of specified time units in the delay. (Wait Value is combined with Units below to get the actual duration. For example, 3 Seconds or 3000 Milliseconds.)

The value of Wait Value may be an Input/Memory/Output tag, a literal numeric value, or some other logical expression that is evaluated every time the block is executed.

To configure the Wait Value parameter, click the  button to open a standard [Build Argument](#) dialog.

- **Units** – The time units in which the Wait Value above is counted. Available units include Milliseconds, Seconds, and Minutes.

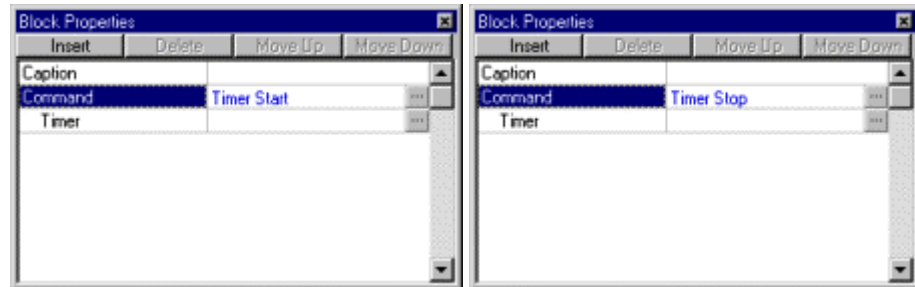
To configure the Units parameter, click the  button and select a unit from the drop-down menu.

B.2 Timer Commands

Timer commands (in a **Process block**) are used to start, stop, and reset **Timers**. (For more information on defining Timers in Logic memory, see page 117.)

B.2.1 Timer Start and Timer Stop


These commands can be selected from the **Timer commands** list.



When used in a Flow Chart, the Timer Start and Timer Stop commands start and stop a specified **Timer**.

Parameters for these commands include:

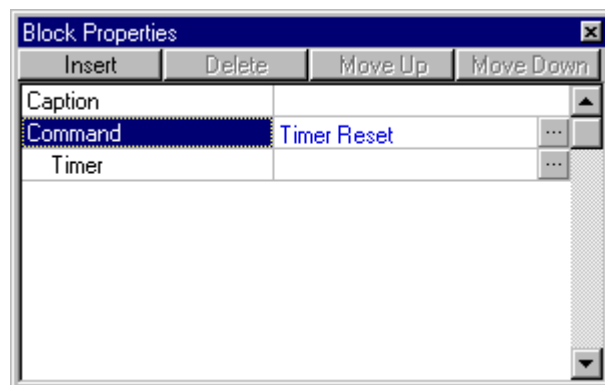
- **Timer** – The Timer to be started or stopped by the command.

To configure the Timer parameter, click the  button to open a standard **Build Timer ID** dialog.

NOTE: If the Timer was previously stopped, the Timer Start command resumes the Timer from its last value. To start the Timer from 0, it must first be reset using the Timer Reset command.

B.2.2 Timer Reset


This command can be selected from the **Timer commands** list.



When used in a Flow Chart, the Timer Reset command resets the specified **Timer** to 0 msec.

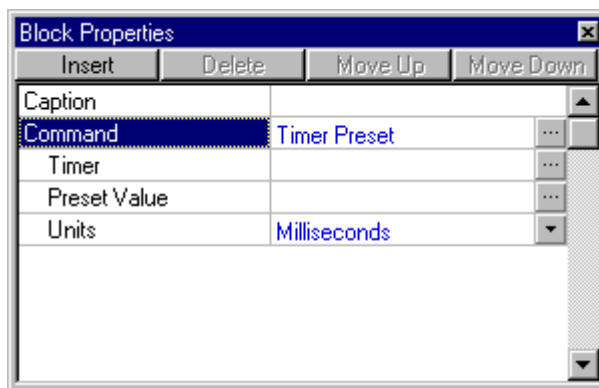
Parameters for this command include:

- **Timer** – The Timer to be reset by the command.

To configure the Timer parameter, click the  button to open a standard **Build Timer ID** dialog.

B.2.3 Timer Preset


This command can be selected from the **Timer commands** list.



When used in a Flow Chart, the Timer Preset command assigns a new Preset value to a specified **Timer**. This value overwrites whatever Preset was set when the Timer was originally **defined** in Logic Memory, and it is retained until the program is restarted or until the Preset is overwritten again by another Timer Preset command.


Parameters for this command include:

- **Timer** – The Timer for which the Preset is to be defined.


To configure the Timer parameter, click the  button to open a standard **Build Timer ID** dialog.

- **Preset Value** – The number of specified time units in the Preset. (Preset Value is combined with Units below to get the actual Preset. For example, 3 Seconds or 3000 Milliseconds.)

The value of Preset Value may be an Input/Memory/Output tag, a literal numeric value, or some other logical expression that is evaluated every time the block is executed.

To configure the Preset Value parameter, click the  button to open a **Build Preset Value** dialog.

- **Units** – The time units in which the Preset Value above is counted. Available units include Milliseconds, Seconds, and Minutes.

To configure the Units parameter, click the  button and select a unit from the drop-down menu.

NOTE: Although other time units can be selected when using this command, the Timer itself still counts in milliseconds during runtime. (The Preset value that is entered is recalculated in milliseconds.)

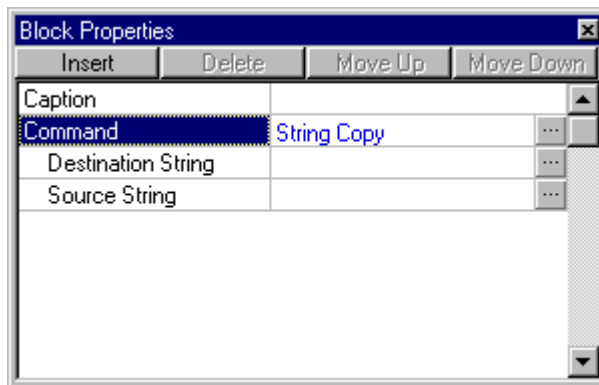
B.3 String Commands

String commands (in a [Process block](#)) are used to manipulate strings and [String tags](#).

NOTE: All String outputs are NULL-terminated.

B.3.1 String Copy

This command can be selected from the [String commands](#) list.



When used in a Flow Chart, the String Copy commands directly copies one String into another.

Parameters for this command include:

- **Destination String** – The String tag to which the Source String will be copied.

To configure the Destination String parameter, click the button to open a standard [Select String Tag](#) dialog.

NOTE: If the defined length of the Destination String is not long enough to receive the result of the command, then the output will be truncated. No error will be generated.

- **Source String** – The string to be copied into the Destination String.

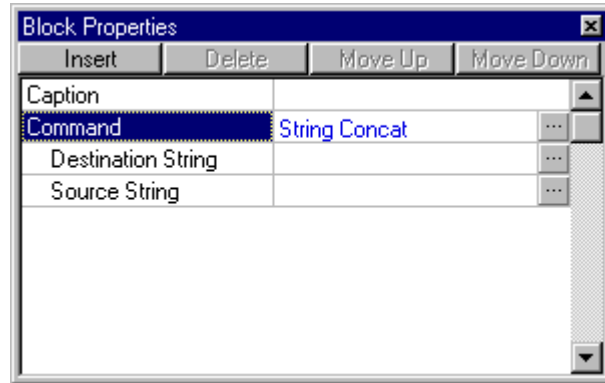
The value of Source String may be a String tag, a literal string enclosed in single quotes (' '), or some other logical expression that is evaluated every time the block is executed.

To configure the Source String parameter, click the button to open a standard [Build String Argument](#) dialog.

NOTE: Source String is not modified by the execution of this command.

B.3.2 String Concat

This command can be selected from the [String commands](#) list.



When used in a Flow Chart, the String Concat command appends one string to the end of another.

Parameters for this command include:

- **Destination String** – The string onto which the Source String will be appended. Also, the String tag to which the output will be written. (Essentially, old Destination String + Source String = new Destination String.)

To configure the Destination String parameter, click the button to open a standard [Select String Tag](#) dialog.

NOTE: If the defined length of the Destination String is not long enough to receive the result of the command, then the output will be truncated. No error will be generated.

- **Source String** – The string to be appended to the end of the Destination String.

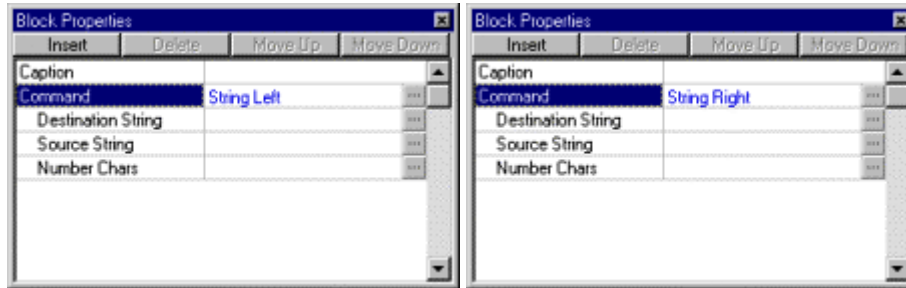
The value of Source String may be a String tag, a literal string enclosed in single quotes (' '), or some other logical expression that is evaluated every time the block is executed.

To configure the Source String parameter, click the button to open a standard [Build String Argument](#) dialog.

NOTE: Source String is not modified by the execution of this command.

B.3.3 String Left and String Right

These commands can be selected from the [String commands](#) list.



When used in a Flow Chart, the String Left and String Right commands extract a sub-string of a specified length from a given string. The String Left command extracts from the left side of the string. The String Right command extracts from the right side of the string.

Parameters for these commands include:

- **Destination String** – The String tag to which the extracted sub-string will be written.

To configure the Destination String parameter, click the button to open a standard [Select String Tag](#) dialog.

NOTE: If the defined length of the Destination String is not long enough to receive the result of the command, then the output will be truncated. No error will be generated.

- **Source String** – The string from which the sub-string will be extracted.

The value of Source String may be a String tag, a literal string enclosed in single quotes (' '), or some other logical expression that is evaluated every time the block is executed.

To configure the Source String parameter, click the button to open a standard [Build String Argument](#) dialog.

NOTE: Source String is not modified by the execution of this command.

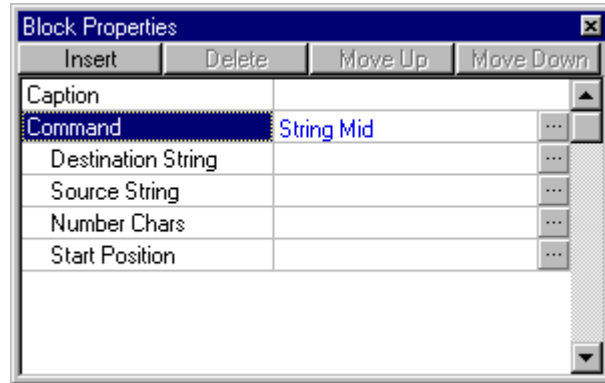
- **Number Chars** – The length of the sub-string, in characters, to be extracted.

The value of Number Chars may be an Input/Memory/Output tag, a literal numeric value, or some other logical expression that is evaluated every time the block is executed.

To configure the Number Chars parameter, click the button to open a standard [Build Argument](#) dialog.

B.3.4 String Mid

This command can be selected from the [String commands](#) list.



When used in a Flow Chart, the String Mid command extracts a sub-string, of a specified length and starting from a specified position, from a given string.

Parameters for this command include:

- **Destination String** – The String tag to which the extracted sub-string will be written.

To configure the Destination String parameter, click the button to open a standard [Select String Tag](#) dialog.

NOTE: If the defined length of the Destination String is not long enough to receive the result of the command, then the output will be truncated. No error will be generated.

- **Source String** – The string from which the sub-string will be extracted.

The value of Source String may be a String tag, a literal string enclosed in single quotes (' '), or some other logical expression that is evaluated every time the block is executed.

To configure the Source String parameter, click the button to open a standard [Build String Argument](#) dialog.

NOTE: Source String is not modified by the execution of this command.


- **Number Chars** – The length of the sub-string, in characters, to be extracted.

The value of Number Chars may be an Input/Memory/Output tag, a literal numeric value, or some other logical expression that is evaluated every time the block is executed.

To configure the Number Chars parameter, click the button to open a standard [Build Argument](#) dialog.

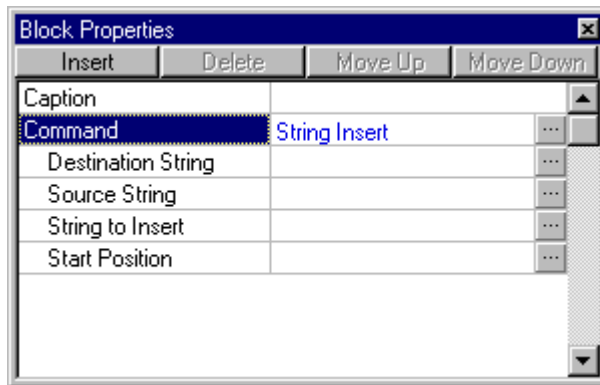
- **Start Position** – The starting position from which the sub-string will be extracted. The first character of the Source String is equivalent to a Start Position of 1.

The value of Start Position may be an Input/Memory/Output tag, a literal numeric value, or some other logical expression that is evaluated every time the block is executed.

To configure the Start Position parameter, click the  button to open a standard **Build Argument** dialog.

B.3.5 String Insert


This command can be selected from the **String commands** list.



When used in a Flow Chart, the String Insert command inserts one string into another, starting at a specified position, and writes the result to separate String tag.

Parameters for this command include:

- **Destination String** – The String tag to which the result of the command will be written.

To configure the Destination String parameter, click the  button to open a standard **Select String Tag** dialog.

NOTE: If the defined length of the Destination String is not long enough to receive the result of the command, then the output will be truncated. No error will be generated.

- **Source String** – The string into which the String to Insert will be inserted.


The value of Source String may be a String tag, a literal string enclosed in single quotes (' '), or some other logical expression that is evaluated every time the block is executed.

To configure the Source String parameter, click the  button to open a standard **Build String Argument** dialog.

NOTE: Source String is not modified by the execution of this command.

- **String to Insert** – The string that will be inserted into the Source String.

The value of String to Insert may be a String tag, a literal string enclosed in single quotes (' '), or some other logical expression that is evaluated every time the block is executed.

To configure the Source String parameter, click the  button to open a standard **Build String Argument** dialog.

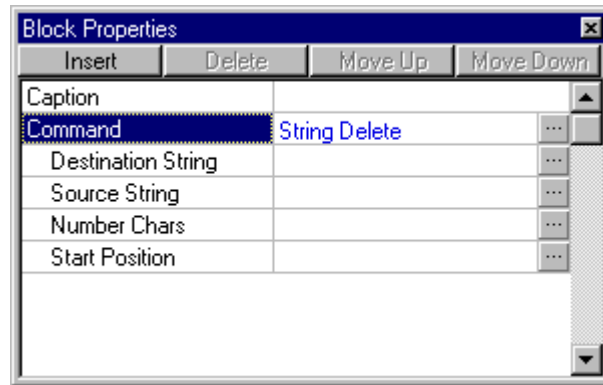
- **Start Position** – The starting position at which the String to Insert will be inserted. The first character of the Source String is equivalent to a Start Position of 1.

The value of Start Position may be an Input/Memory/Output tag, a literal numeric value, or some other logical expression that is evaluated every time the block is executed.

To configure the Start Position parameter, click the  button to open a standard **Build Argument** dialog.

B.3.6 String Delete


This command can be selected from the **String commands** list.



When used in a Flow Chart, the String Delete command deletes a specified number of characters from a given string and writes the result to separate String tag. This is different from the String Left, String Right, and String Mid commands above; the output is the remainder of the given string rather than the characters that were deleted from it.

Parameters for this command include:


- **Destination String** – The String tag to which the result of the command will be written.

To configure the Destination String parameter, click the  button to open a standard **Select String Tag** dialog.

NOTE: If the defined length of the Destination String is not long enough to receive the result of the command, then the output will be truncated. No error will be generated.

- **Source String** – The string from which the specified characters will be deleted.


The value of Source String may be a String tag, a literal string enclosed in single quotes (' '), or some other logical expression that is evaluated every time the block is executed.

To configure the Source String parameter, click the  button to open a standard **Build String Argument** dialog.

NOTE: Source String is not modified by the execution of this command.


- **Number Chars** – The number of characters to be deleted from the Source String.

The value of Number Chars may be an Input/Memory/Output tag, a literal numeric value, or some other logical expression that is evaluated every time the block is executed.

To configure the Number Chars parameter, click the  button to open a standard **Build Argument** dialog.

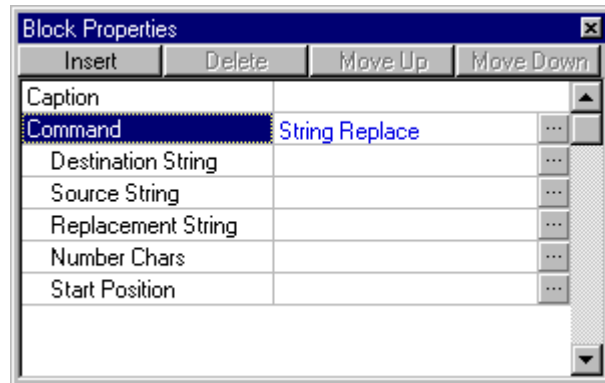
- **Start Position** – The starting position from which the specified characters will be deleted. The first character of the Source String is equivalent to a Start Position of 1.

The value of Start Position may be an Input/Memory/Output tag, a literal numeric value, or some other logical expression that is evaluated every time the block is executed.

To configure the Start Position parameter, click the  button to open a standard **Build Argument** dialog.

B.3.7 String Replace

This command can be selected from the [String commands](#) list.



When used in a Flow Chart, the String Replace command replaces a specified number of characters in a given string with the contents of another string. The result is written to a separate String tag. This is different from the String Insert command above; the specified characters are completely overwritten rather than simply displaced.

Parameters for this command include:

- **Destination String** – The String tag to which the result of the command will be written.

To configure the Destination String parameter, click the button to open a standard [Select String Tag](#) dialog.

NOTE: If the defined length of the Destination String is not long enough to receive the result of the command, then the output will be truncated. No error will be generated.

- **Source String** – The string in which the specified characters will be replaced.


The value of Source String may be a String tag, a literal string enclosed in single quotes (' '), or some other logical expression that is evaluated every time the block is executed.

To configure the Source String parameter, click the button to open a standard [Build String Argument](#) dialog.

NOTE: Source String is not modified by the execution of this command.


- **Replacement String** – The string which will replace the specified characters in the Source String.

The value of Replacement String may be a String tag, a literal string enclosed in single quotes (' '), or some other logical expression that is evaluated every time the block is executed.

To configure the Source String parameter, click the  button to open a standard [Build String Argument](#) dialog.


- **Number Chars** – The number of characters to be replaced in the Source String.

The value of Number Chars may be an Input/Memory/Output tag, a literal numeric value, or some other logical expression that is evaluated every time the block is executed.

To configure the Number Chars parameter, click the  button to open a standard [Build Argument](#) dialog.

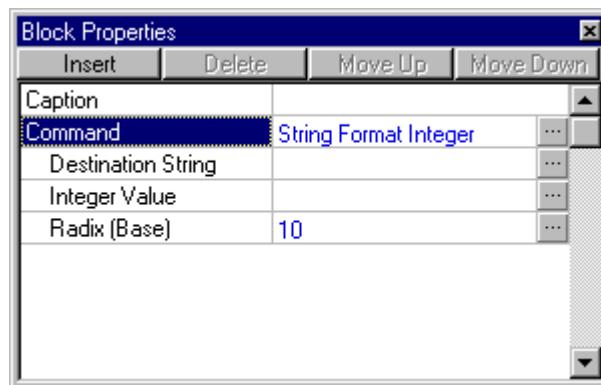
- **Start Position** – The starting position after which the specified characters will be replaced. The first character of the Source String is equivalent to a Start Position of 1.

The value of Start Position may be an Input/Memory/Output tag, a literal numeric value, or some other logical expression that is evaluated every time the block is executed.

To configure the Start Position parameter, click the  button to open a standard [Build Argument](#) dialog.

B.3.8 String Format Integer


This command can be selected from the [String commands](#) list.



When used in a Flow Chart, the String Format Integer command formats an integer value as a string using the specified radix (base). The result is written to a separate String tag. This is especially useful for converting Input, Memory, and Output tags to equivalent String tags for further manipulation.

Parameters for this command include:


- **Destination String** – The String tag to which the result of the command will be written.

To configure the Destination String parameter, click the  button to open a standard [Select String Tag](#) dialog.

NOTE: If the defined length of the Destination String is not long enough to receive the result of the command, then the output will be truncated. No error will be generated.


- **Integer Value** – The integer value to be formatted.

The value of Integer Value may be an Input/Memory/Output tag, a literal numeric value, or some other logical expression that is evaluated every time the block is executed.

To configure the Number Chars parameter, click the  button to open a standard **Build Argument** dialog.

- **Radix (Base)** – The base system by which the Integer Value will be formatted. For example, a Radix of 10 will format the Integer Value as a decimal, while a Radix of 16 will format the Integer Value as a hexadecimal.

The value of Radix (Base) may be an Input/Memory/Output tag, a literal numeric value, or some other logical expression that is evaluated every time the block is executed.

To configure the Start Position parameter, click the  button to open a standard **Build Argument** dialog.

B.4 Diagnostics Commands

Diagnostics commands (in a **Process block**) are used to read from and write to the diagnostic items that are associated with each Input, Memory, and Output tag.

About Diagnostic Items

Every Input, Memory, and Output tag has associated with it three metadata registers, plus a fourth register for Input and Output tags only. These registers are allocated automatically when each tag is defined in Logic Memory, and they can be manipulated independently of the tag's primary value.

The four metadata registers – also called “diagnostic items” – are:

- **Diag Fault Bit** – This item is used simply to indicate whether a fault condition exists on the specified tag. The item occupies a single bit, which can be toggled between 0 and 1.
- **Diag Fault Level** – This item can be used to indicate escalating levels of fault on the specified tag. The item occupies a 32-bit register.
- **Diag Status Code** – This item can be used to record more complex error codes on the specified tag. The item occupies a 32-bit register.
- **Error Status Bit** – This item, available only on Input and Output tags, is a special flag that is used to notify the program when an I/O error has been encountered (see below). The item occupies a single bit, which can be toggled between 0 and 1.

PointeControl itself does not utilize the Diag Fault Bit and Diag Fault Level items; you are free to implement your own custom routines that write to and read from these items, using the Diagnostics commands described below as well as the Diag Fault Bit Test option of decision-type blocks (**If/Then**, **Repeat/Until**, and **While Loop**).

However, PointeControl *does* internally utilize the Diag Status Code and Error Status Bit items. For each **I/O point** on the Pointe Controller unit (and on any connected **OptiLogic RTUs**), the controller automatically notes I/O errors encountered during runtime and logs them on the Input or Output tag associated with the affected point. When an error is encountered, the tag's Error Status Bit is set to 1 and the error code is copied to the tag's Diag Status Code. Possible error codes include:

CODE	DESCRIPTION
0x00000200 (512)	I/O module or operator panel not found
0x00000400 (1024)	OptiLogic RTU not found
0x00004000 (16384)	No value read yet
0x00040000 (262144)	System error
0x00200000 (2097152)	Communication error

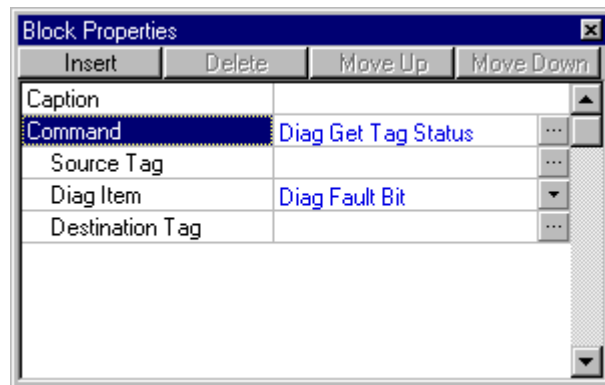
For all other errors, please contact Nematron Technical Support.

NOTE: The Pointe Controller unit updates a given tag's Error Status Bit and Diag Status Code only when a new error is encountered at the tag's associated I/O point. As such, the values currently stored in the items should always be handled as merely the last error encountered, not as an ongoing error condition. To actually clear the items, use the **Diag Clear Tag Status** command.

Also, clearing one item does *not* clear the other. Each item must be cleared individually using separate commands.

B.4.1 Diag Get Tag Status

This command can be selected from the **Diagnostics commands** list.



When used in a Flow Chart, the Diag Get Tag Status command retrieves the current value of any diagnostic item on a specified tag.

Parameters for this command include:


- **Source Tag** – The Input, Output, or Memory tag for which the diagnostic item will be retrieved.

To configure the Source Tag parameter, click the button to open a standard **Select Tag** dialog.

- **Diag Item** – The type of diagnostic item that will be retrieved by the command. Available items include Diag Fault Bit, Diag Fault Level, Diag Status Code, and Error Status Bit (input and Output tags only).

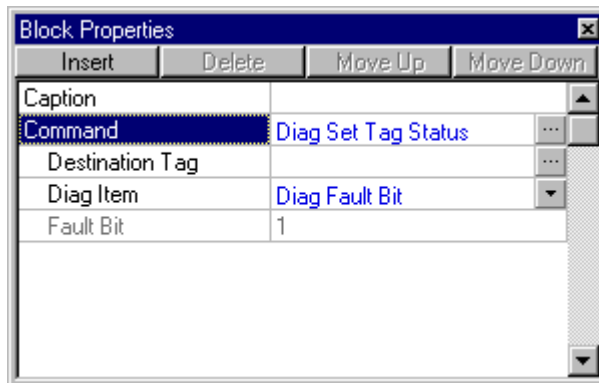
To configure the Diag Item parameter, click the button and select an item from the drop-down menu.

- **Destination Tag** – The Memory or Output tag to which the retrieved value will be saved. If the Diag Fault Bit or Error Status Bit will be retrieved, then the value should be saved to a Bit tag (%MX or %QX). If the Diag Fault Level or Diag Status Code will be retrieved, then the value should be saved to a 32-bit Unsigned tag (%MUD or %QUD).

To configure the Destination Tag parameter, click the  button to open a standard [Select Tag](#) dialog.

B.4.2 Diag Set Tag Status


This command can be selected from the [Diagnostics commands](#) list.




When used in a Flow Chart, the Diag Set Tag Status command assigns a new value to any diagnostic item on a specified tag.

Parameters for this command include:

- **Destination Tag** – The Input, Output, or Memory tag on which the diagnostic item will be assigned.

To configure the Destination Tag parameter, click the  button to open a standard [Select Tag](#) dialog.

- **Diag Item** – The type of diagnostic item that will be assigned by the command. Available items include Diag Fault Bit, Diag Fault Level, Diag Status Code, and Error Status Bit (input and Output tags only).

To configure the Diag Item parameter, click the  button and select an item from the drop-down menu.

- The third parameter varies depending on which Diag Item is selected:
 - If Diag Fault Bit is selected, then the third parameter is **Fault Bit**. The parameter is not configurable since it simply toggles the bit from 0 to 1. (To reset the bit to 0, use the Diag Clear Tag Status command described below.)
 - If Diag Fault Level is selected, then the third parameter is **New Fault Level**. You can define your own fault levels in your diagnostic routines.

The value of Diag Fault Level may be an Input/Memory/Output tag, a literal numeric value, or some other logical expression that is evaluated every time the block is executed.

To configure the Diag Fault Level parameter, click the  button to open a standard **Build Argument** dialog.

- If Diag Status Code is selected, then the third parameter is **New Status Code**. You can define your own status codes in your diagnostic routines, but PointeControl does automatically record I/O errors (see above).

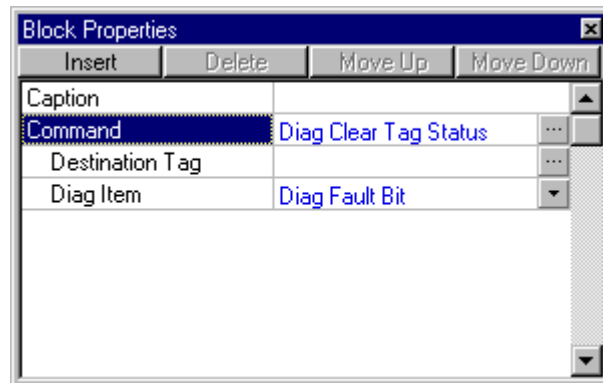
The value of Diag Status Code may be an Input/Memory/Output tag, a literal numeric value, or some other logical expression that is evaluated every time the block is executed.

To configure the Diag Status Code parameter, click the  button to open a standard **Build Argument** dialog.

- If Error Status Bit is selected, then the third parameter is **Error Status Bit**. The parameter is not configurable since it simply toggles the bit from 0 to 1. (To reset the bit to 0, use the Diag Clear Tag Status command described below.)

B.4.3 Diag Clear Tag Status


This command can be selected from the **Diagnostics commands** list.




When used in a Flow Chart, the Diag Clear Tag Status command clears the current value of any diagnostic item (resets to 0) on a specified tag.

Parameters for this command include:

- **Destination Tag** – The Input, Output, or Memory tag on which the diagnostic item will be cleared.

To configure the Destination Tag parameter, click the  button to open a standard **Select Tag** dialog.

- **Diag Item** – The type of diagnostic item that will be cleared by the command. Available items include Diag Fault Bit, Diag Fault Level, Diag Status Code, and Error Status Bit (input and Output tags only).

To configure the Diag Item parameter, click the  button and select an item from the drop-down menu.

B.5 Serial Commands

The Serial commands are used to initiate serial communications through the Pointe Controller I/O system.

NOTE: Read/Write commands do not normally block or wait. Read gets all bytes which have been buffered on the port; write queues the byte(s) for output. The command does not wait until the characters are physically transmitted.

About COM Ports

The Pointe Controller unit has one serial port built into the base unit itself, and more ports can be added by installing a [Dual Port RS232 Serial](#) module (OL2602) in one of the unit's slots.

The built-in port is COM Port 0. The rest of the port numbers are determined by which slot the OL2602 module is installed in:

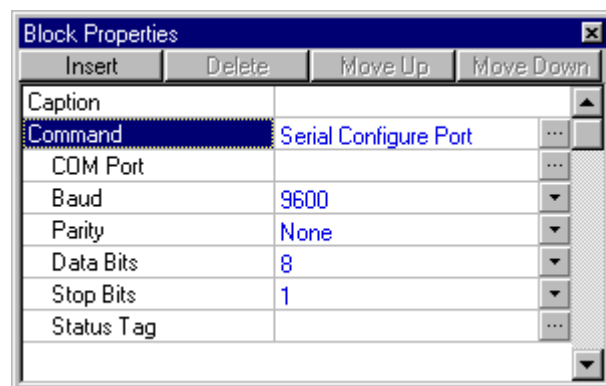
Controller Slot	1	2	3	4	5	6	7	8
Top COM Port	1	3	5	7	9	11	13	15
Bottom COM Port	2	4	6	8	10	12	14	16

As such, if you had an OL2602 module installed in slot 4, then the top port would be COM Port 7 and the bottom port would be COM Port 8.

NOTE: If you attempt to use a COM Port number that does not exist – because no OL2602 module is installed in that slot – then the Pointe Controller will return a Status Tag error.

B.5.1 Serial Configure Port

This command can be selected from the [Serial commands](#) list.




When used in a Flow Chart, the Serial Configure Port command configures the specified COM port for serial communications.

NOTE: This command also **enables** (opens) the specified port.

Parameters for this command include:

- **COM Port** – Specify which serial communications port to configure. For more information on how the ports are numbered, see page 321.


The value of COM Port may be an Input/Memory/Output tag, a literal numeric value, or some other logical expression that is evaluated every time the block is executed.

To configure the COM Port parameter, click the  button to open a standard **Build Argument** dialog.


- **Baud** – 300, 1200, 2400, 4800, 9600, or 19200 baud; default is 9600.

To configure the Baud parameter, click the  button and select a setting from the drop-down menu.


- **Parity** – None, Even, or Odd parity; default is None.

To configure the Parity parameter, click the  button and select a setting from the drop-down menu.

- **Data Bits** – 7 or 8 data bits; default is 8.


To configure the Data Bits parameter, click the  button and select a setting from the drop-down menu.

- **Stop Bits** – 1, 1.5, or 2 stop bits; default is 1.

To configure the Stop Bits parameter, click the  button and select a setting from the drop-down menu.

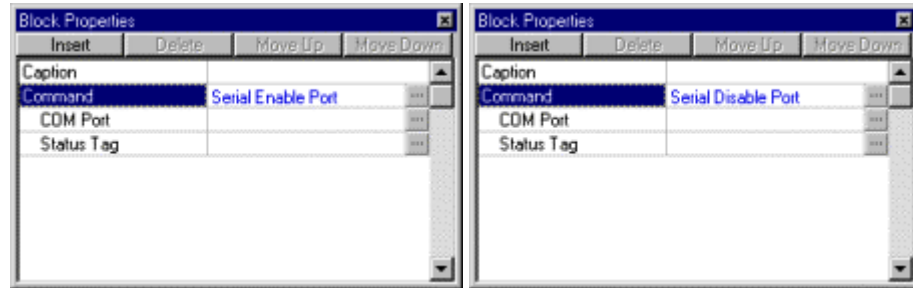
- **Status Tag** – The Serial Configure Port command checks for a communication error at the time of command execution. The Status Tag is the tag to which the error code, if any, is saved. For more information on interpreting error codes, see **“Serial Get Comm Errors”** on page 328.

The value of Status Tag may be any 16-bit Unsigned tag (%IUW, %MUW, or %QUW).

To configure the Status Tag parameter, click the  button to open a standard **Select Tag** dialog.

B.5.2 Serial Enable Port and Serial Disable Port

These commands can be selected from the [Serial commands](#) list.




When used in a Flow Chart, the Serial Enable Port and Serial Disable Port commands open and close the specified COM port, respectively.

Parameters for these commands include:


- **COM Port** – Specify which serial communications port to enable (open) or disable (close). For more information on how the ports are numbered, see page 321.

The value of COM Port may be an Input/Memory/Output tag, a literal numeric value, or some other logical expression that is evaluated every time the block is executed.

To configure the COM Port parameter, click the  button to open a standard [Build Argument](#) dialog.

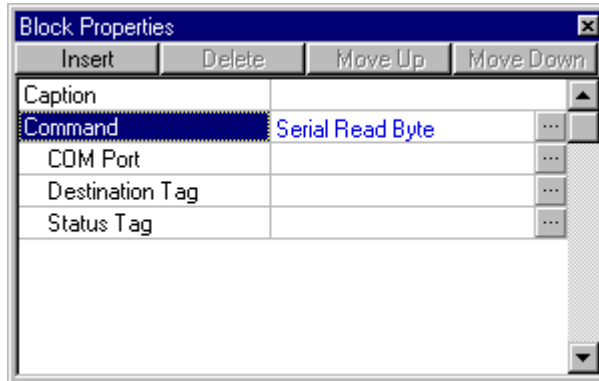
- **Status Tag** – The Serial Enable and Serial Disable commands check for a communication error at the time of command execution. The Status Tag is the tag to which the error code, if any, is saved. For more information on interpreting error codes, see [“Serial Get Comm Errors”](#) on page 328.

The value of Status Tag may be any 16-bit Unsigned tag (%IUW, %MUW, or %QUW).

To configure the Status Tag parameter, click the  button to open a standard [Select Tag](#) dialog.

B.5.3 Serial Read Byte

This command can be selected from the [Serial commands](#) list.




When used in a Flow Chart, the Serial Read Byte command reads a single byte/character from the specified COM port and saves it to a Logic Memory tag.

NOTE: If you need to convert the read byte into an ASCII character, you can use the [Integer to Character](#) ladder block.

Parameters for this command include:


- **COM Port** – Specify which serial communications port to read from. For more information on how the ports are numbered, see page 321.

The value of COM Port may be an Input/Memory/Output tag, a literal numeric value, or some other logical expression that is evaluated every time the block is executed.

To configure the COM Port parameter, click the  button to open a standard [Build Argument](#) dialog.


- **Destination Tag** – The tag to which the read byte will be saved.

The value of Destination Tag may be any 8-bit Unsigned tag (%IUB, %MUB, or %QUB).

To configure the Destination Tag parameter, click the  button to open a standard [Select Tag](#) dialog.

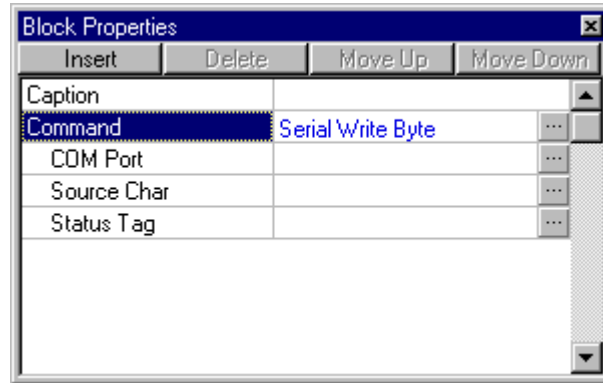
- **Status Tag** – The Serial Read Byte command checks for a communication error at the time of command execution. The Status Tag is the tag to which the error code, if any, is saved. For more information on interpreting error codes, see [“Serial Get Comm Errors”](#) on page 328.

The value of Status Tag may be any 16-bit Unsigned tag (%IUW, %MUW, or %QUW).

To configure the Status Tag parameter, click the  button to open a standard [Select Tag](#) dialog.

B.5.4 Serial Write Byte

This command can be selected from the [Serial commands](#) list.



When used in a Flow Chart, the Serial Write Byte command writes a single byte/character (as defined in the Build Argument dialog) to the specified COM port.

NOTE: String characters are automatically converted into their equivalent ASCII values.

Parameters for this command include:

- **COM Port** – Specify which serial communications port to write to. For more information on how the ports are numbered, see page 321.

The value of COM Port may be an Input/Memory/Output tag, a literal numeric value, or some other logical expression that is evaluated every time the block is executed.

To configure the COM Port parameter, click the button to open a standard [Build Argument](#) dialog.


- **Source Char** – The byte/character that will be written to the COM port.

The value of Source Char may be an Input/Memory/Output tag, a String variable, a literal numeric value, a literal string enclosed in single quotes (' '), or some other logical expression that is evaluated every time the block is executed. If the given value is more than one byte long, then only the first byte will be written.

To configure the Source Char parameter, click the button to open a standard [Build Argument](#) dialog.

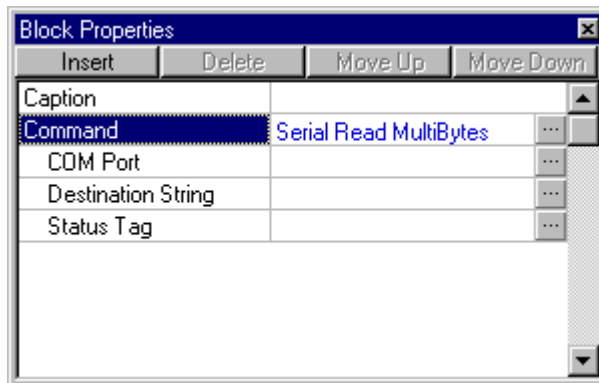
- **Status Tag** – The Serial Write Byte command checks for a communication error at the time of command execution. The Status Tag is the tag to which the error code, if any, is saved. For more information on interpreting error codes, see [“Serial Get Comm Errors”](#) on page 328.

The value of Status Tag may be any 16-bit Unsigned tag (%IUW, %MUW, or %QUW).

To configure the Status Tag parameter, click the  button to open a standard [Select Tag](#) dialog.

B.5.5 Serial Read MultiBytes

This command can be selected from the [Serial commands](#) list.




When used in a Flow Chart, the Serial Read MultiBytes command reads a string from the specified COM port and saves it to a [String](#) variable in Logic Memory. The command will read up to defined Element Length of the variable and will not block until the variable is full.

NOTE: The read bytes will be automatically converted into a string, regardless of how they were originally sent.

Parameters for this command include:

- **COM Port** – Specify which serial communications port to read from. For more information on how the ports are numbered, see page 321.

The value of COM Port may be an Input/Memory/Output tag, a literal numeric value, or some other logical expression that is evaluated every time the block is executed.


To configure the COM Port parameter, click the  button to open a standard [Build Argument](#) dialog.

- **Destination String** – The String variable to which the read bytes will be saved.

To configure the Destination String parameter, click the  button to open a standard [Select String Tag](#) dialog.

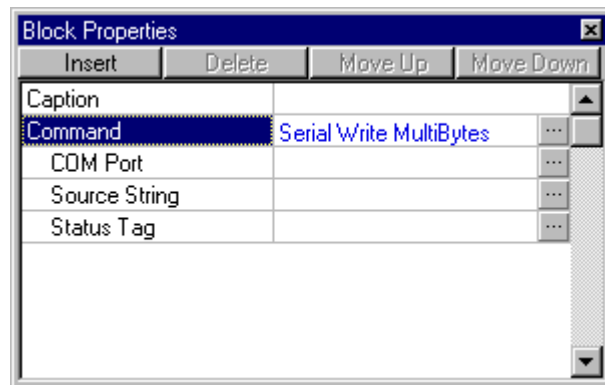
- **Status Tag** – The Serial Read MultiBytes command checks for a communication error at the time of command execution. The Status Tag is the tag to which the error code, if any, is saved. For more information on interpreting error codes, see [“Serial Get Comm Errors”](#) on page 328.

The value of Status Tag may be any 16-bit Unsigned tag (%IUW, %MUW, or %QUW).

To configure the Status Tag parameter, click the  button to open a standard [Select Tag](#) dialog.

B.5.6 Serial Write MultiBytes

This command can be selected from the [Serial commands](#) list.




When used in a Flow Chart, the Serial Write MultiBytes command writes a string (as defined in the Build String Argument dialog) to the specified COM port. It will write only up to the actual length of the given string.

Parameters for this command include:


- **COM Port** – Specify which serial communications port to write to. For more information on how the ports are numbered, see page 321.

The value of COM Port may be an Input/Memory/Output tag, a literal numeric value, or some other logical expression that is evaluated every time the block is executed.

To configure the COM Port parameter, click the  button to open a standard [Build Argument](#) dialog.


- **Source String** – The string that will be written to the COM port.

The value of Source String may be a String variable, a literal string enclosed in single quotes (' '), or some other logical expression that is evaluated every time the block is executed.

To configure the Source String parameter, click the  button to open a standard [Build String Argument](#) dialog.

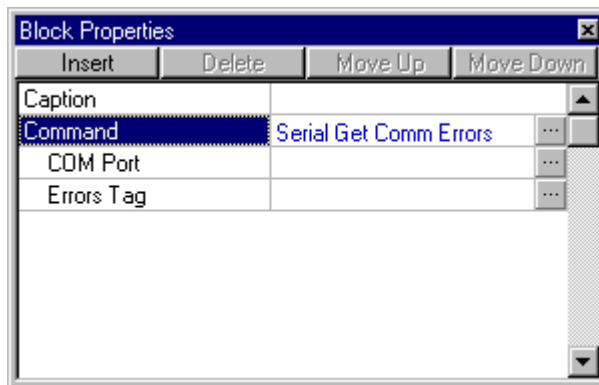
- **Status Tag** – The Serial Write MultiBytes command checks for a communication error at the time of command execution. The Status Tag is the tag to which the error code, if any, is saved. For more information on interpreting error codes, see [“Serial Get Comm Errors”](#) on page 328.

The value of Status Tag may be any 16-bit Unsigned tag (%IUW, %MUW, or %QUW).

To configure the Status Tag parameter, click the  button to open a standard [Select Tag](#) dialog.

B.5.7 Serial Get Comm Errors

This command can be selected from the [Serial commands](#) list.




When used in a Flow Chart, the Serial Get Comm Errors command simply checks the specified COM port for communication errors.

Parameters for this command include:


- **COM Port** – Specify which serial communications port to check. For more information on how the ports are numbered, see page 321.

The value of COM Port may be an Input/Memory/Output tag, a literal numeric value, or some other logical expression that is evaluated every time the block is executed.

To configure the COM Port parameter, click the  button to open a standard [Build Argument](#) dialog.

- **Errors Tag** – The tag to which the error code, if any, is saved.

The value of Errors Tag may be any 16-bit Unsigned tag (%IUW, %MUW, or %QUW).

To configure the Errors Tag parameter, click the  button to open a standard [Select Tag](#) dialog.

Error Codes

To interpret a serial communication error code, get the value that was saved to the Status/Errors Tag and check it against the table below:

VALUE	NAME	DESCRIPTION
0x0000 (0)	n/a	No error; good communication
0x0001 (1)	PARITY_ERROR	OptiLogic parity error
0x0002 (2)	FRAMING_ERROR	OptiLogic framing error
0x0004 (4)	OVERRUN_ERROR	OptiLogic overrun error
0x0010 (16)	READ_ERROR	Serial read error
0x0020 (32)	WRITE_ERROR	Serial write error
0x0040 (64)	BAD_CONFIG_ERROR	Invalid configuration error
0x0080 (128)	BAD_PORT_ERROR	Non-existent serial port error
0x0100 (256)	OVERFLOW_ERROR	Buffer overflow error
0x8000 (32768)	NO_DATA	No data available on read

For all other errors, please contact Nematron Technical Support.

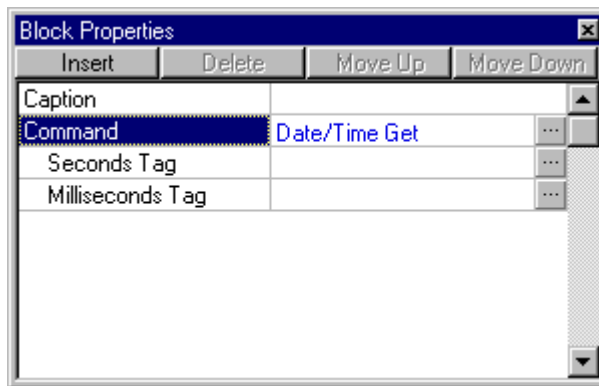
B.6 Date/Time Commands

Date/Time commands (in a **Process block**) are used to retrieve the system time on the Pointe Controller unit.

NOTE: The controller's internal clock is updated whenever a compiled project is downloaded from your PC to the controller. The time is taken from your PC's system time.

B.6.1 Date/Time Get

This command can be selected from the **Date/Time commands** list.



When used in a Flow Chart, the Date/Time Get command retrieves the current system time on the Pointe Controller unit. The time is expressed as the number of whole seconds and remaining milliseconds elapsed since January 1, 1970. The Seconds and Milliseconds values are both returned as 32-bit unsigned integers.

Parameters for this command include:

- **Seconds Tag** – The Logic Memory tag to which the number of seconds will be written. The tag should be a 32-bit Unsigned Input, Memory, or Output tag.

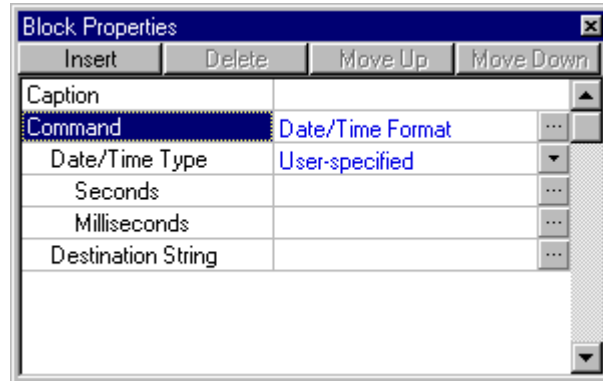
To configure the Seconds Tag parameter, click the button to open a standard **Select Tag** dialog.

- **Milliseconds Tag** – The Logic Memory tag to which the number of remaining milliseconds will be written. The tag should be a 32-bit Unsigned Input, Memory, or Output tag.

To configure the Milliseconds Tag parameter, click the button to open a standard **Select Tag** dialog.

B.6.2 Date/Time Format


This command can be selected from the [Date/Time commands](#) list.




When used in a Flow Chart, the Date/Time Format command formats a given time as a string and writes the result to a String tag. The format of the string is <ddd mmm DD HH:MM:SS.sss YYYY>.

Parameters for this command include:

- **Date/Time Type** – The time to be formatted, either the current system time or some user-specified time. If the current system time is selected, then it will be retrieved automatically when the block is executed. If a user-specified time is selected, then the Seconds and Milliseconds parameters below must also be configured.


To configure the Date/Time Type parameter, click the  button and select a type from the drop-down menu.

- **Seconds** – The number of whole seconds elapsed since January 1, 1970.
The value of Seconds may be an Input/Memory/Output tag, a literal numeric value, or some other logical expression that is evaluated every time the block is executed.

To configure the Seconds parameter, click the  button to open a standard [Build Argument](#) dialog.

- **Milliseconds** – The number of remaining milliseconds (total time elapsed minus whole seconds).

The value of Milliseconds may be an Input/Memory/Output tag, a literal numeric value, or some other logical expression that is evaluated every time the block is executed.

To configure the Milliseconds parameter, click the  button to open a standard [Build Argument](#) dialog.

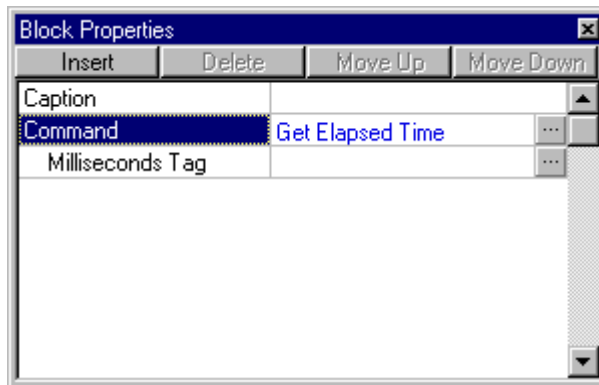
- **Destination String** – The String tag to which the result of the command will be written.

To configure the Destination String parameter, click the  button to open a standard [Select String Tag](#) dialog.

NOTE: If the defined length of the Destination String is not long enough to receive the result of the command, then the output will be truncated. No error will be generated.

B.6.3 Get Elapsed Time


This command can be selected from the [Date/Time commands](#) list.



When used in a Flow Chart, the Get Elapsed Time command retrieves the total time elapsed, in milliseconds, since the Pointe Controller unit was last powered on. The Milliseconds value is returned as a 32-bit unsigned integer.

Parameters for this command include:

- **Milliseconds Tag** – The Logic Memory tag to which the number of milliseconds will be written. The tag should be a 32-bit Unsigned Input, Memory, or Output tag.

To configure the Milliseconds Tag parameter, click the  button to open a standard [Select Tag](#) dialog.

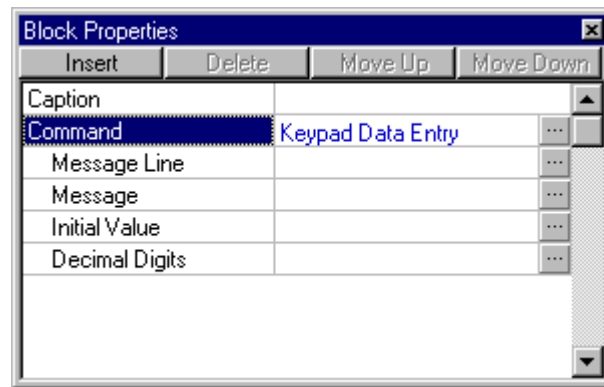
B.7 Operator Panel Commands

Operator Panel commands (in a **Process block**) are used to control the OL3406, OL3420, and OL3850 operator panels. You must have one of these panels connected to your Pointe Controller unit and properly configured in order to use these commands. For more information, see “**Configuring operator panels**” on page 128.

NOTE: The Keypad Data Entry and Arrow Adjust Data Entry commands work only with the OL3850 operator panel. The Button On and Button Off commands work with all three panel models.

B.7.1 Keypad Data Entry

This command can be selected from the **Operator Panel commands** list.



When used in a Flow Chart, the Keypad Data Entry command displays a message prompting the user to enter a value using the panel’s numeric keypad. After the user enters a value and presses the ENTER key, the entered value is saved to the Data Value tag and the Data Available tag is set to 1.

NOTE: This command works only with the **OL3850 operator panel**.

Parameters for this command include:

- **Message Line** – Which line of the operator panel that will be used to display the message. The top line is 1, the bottom line is 2.


The value of Message Line may be an Input/Memory/Output tag, a literal numeric value, or some other logical expression that is evaluated every time the block is executed.

To configure the Message Line parameter, click the  button to open a standard **Build Argument** dialog.

- **Message** – The text of the message, as specified in the Build String Argument dialog. Carets “^” must be included in the message text as a


place holders for the data entry field. A typical message may be 'Enter value: ^^^^^^^', allowing up to seven digits to be entered.

The value of Message may be a String tag, a literal string enclosed in single quotes (' '), or some other logical expression that is evaluated every time the block is executed.

To configure the Message parameter, click the  button to open a standard **Build String Argument** dialog.


- **Initial Value** – The initial value that is displayed in the entry field. It can be any integer or real number, as specified in the Build Argument dialog. The value entered by the user overwrites this value.

The value of Initial Value may be an Input/Memory/Output tag, a literal numeric value, or some other logical expression that is evaluated every time the block is executed.

To configure the Initial Value parameter, click the  button to open a standard **Build Argument** dialog.

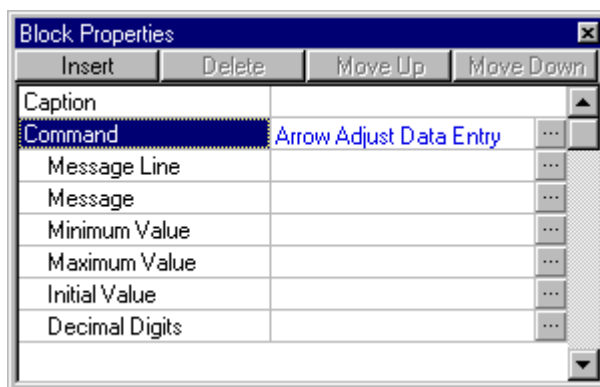
- **Decimal Digits** – The number of decimal places to which the Initial Value is displayed. Integers are padded, reals are truncated.

The value of Decimal Digits may be an Input/Memory/Output tag, a literal numeric value, or some other logical expression that is evaluated every time the block is executed.

To configure the Decimal Digits parameter, click the  button to open a standard **Build Argument** dialog.

B.7.2 Arrow Adjust Data Entry

This command can be selected from the **Operator Panel commands** list.




When used in a Flow Chart, the Arrow Adjust Data Entry command displays a message prompting the user to enter a value using the panel's arrow buttons. After the user enters a value and presses the ENTER key, the entered value is saved to the Data Value tag and the Data Available tag is set to 1.

NOTE: This command works only with the [OL3850 operator panel](#).

Parameters for this command include:


- **Message Line** – Which line of the operator panel that will be used to display the message. The top line is 1, the bottom line is 2.

The value of Message Line may be an Input/Memory/Output tag, a literal numeric value, or some other logical expression that is evaluated every time the block is executed.

To configure the Message Line parameter, click the  button to open a standard [Build Argument](#) dialog.


- **Message** – The text of the message, as specified in the Build Argument dialog. Carets “^” must be included in the message text as a place holders for the data entry field. A typical message may be ‘Enter value: ^^^^^^^’, allowing up to seven digits to be entered.

The value of Message may be a String tag, a literal string enclosed in single quotes (‘ ’), or some other logical expression that is evaluated every time the block is executed.

To configure the Message parameter, click the  button to open a standard [Build String Argument](#) dialog.


- **Minimum Value** – The minimum possible value that can be entered. If the user presses and holds the down arrow, the entry field will stop decrementing at this value.

The value of Minimum Value may be an Input/Memory/Output tag, a literal numeric value, or some other logical expression that is evaluated every time the block is executed.

To configure the Minimum Value parameter, click the  button to open a standard [Build Argument](#) dialog.


- **Maximum Value** – The maximum possible value that can be entered. If the user presses and holds the up arrow, the entry field will stop incrementing at this value.

The value of Maximum Value may be an Input/Memory/Output tag, a literal numeric value, or some other logical expression that is evaluated every time the block is executed.

To configure the Maximum Value parameter, click the  button to open a standard [Build Argument](#) dialog.


- **Initial Value** – The initial value that is displayed in the entry field. It can be any integer or real number, as specified in the Build Argument dialog. The value entered by the user overwrites this value.

The value of Initial Value may be an Input/Memory/Output tag, a literal numeric value, or some other logical expression that is evaluated every time the block is executed.

To configure the Initial Value parameter, click the  button to open a standard [Build Argument](#) dialog.

- **Decimal Digits** – The number of decimal places to which the Initial Value is displayed. Integers are padded, reals are truncated.

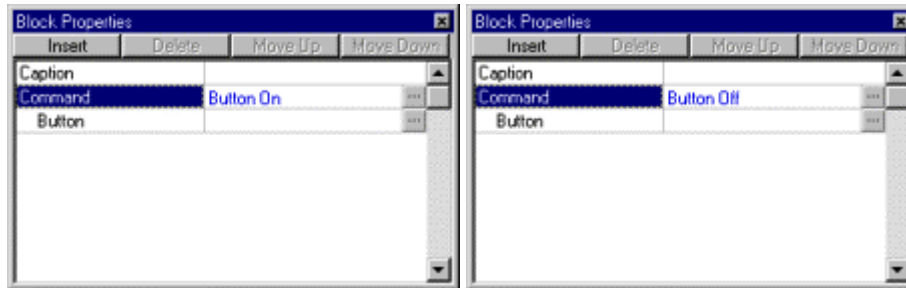
The value of Decimal Digits may be an Input/Memory/Output tag, a literal numeric value, or some other logical expression that is evaluated every time the block is executed.

To configure the Decimal Digits parameter, click the  button to open a standard [Build Argument](#) dialog.

NOTE: In both Data Entry commands, the message is removed from the display as soon as the user presses the ENTER key. It is replaced with the current value of String variable associated with the I/O point.

B.7.3 Button On and Button Off

These commands can be selected from the [Operator Panel commands](#) list.




When used in a Flow Chart, the Button On and Button Off commands force the specified pushbutton on or off, respectively. The pushbutton is specified by the Button parameter, and the buttons on each panel are numerically addressed from left to right. For example, to force the second pushbutton on an OL3850 panel to turn on, you would use the **Button On** command and set the **Button** parameter to 2.

NOTE: The buttons should already be set “alternate action.” For more information, see [“Configuring operator panels”](#) on page 128.

Parameters for this command include:

- **Button** – Which button on the panel that will be forced on/off by the command. The buttons on each panel are numbered from left to right.

The value of Button may be an Input/Memory/Output tag, a literal numeric value, or some other logical expression that is evaluated every time the block is executed.

To configure the Button parameter, click the  button to open a standard [Build Argument](#) dialog.

Appendix C: Ladder Diagram Block Reference

This appendix provides complete descriptions and configuration instructions for all of the Ladder Diagram function blocks that are available in the PointeControl development framework. (For more information on building Ladder Diagrams, see page 163.)

TIP: The information provided in this appendix is also available via the PointeControl Framework online help. To access the help, choose **Contents** from the Framework's **Help** menu.

C.1 Relays and Coils

C.1.1 Normally Open Contact (XIC)

When used in a Ladder Diagram, this block acts as a relay that controls the passing of the rung state from the left to the right. The state is passed only if the value of the associated Bit tag is true or 1.

Select the  tool (from the **Relays and Coils** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

When the input state from the left becomes on, the value of the associated Bit tag is checked. If the tag is true or 1, then the output state to the right is turned on. If the tag is false or 0, then the output state is kept off.

The relay is checked every time the ladder is **scanned**, so long as the input state remains on.

Configuration Reference

Any Bit tag may be associated with the block: %IX, %MX, %QX, or T_DONE.

C.1.2 Normally Closed Contact (XIO)

When used in a Ladder Diagram, this block acts as a relay that controls the passing of the rung state from the left to the right. The state is passed only if the value of the associated Bit tag is false or 0.

Select the  tool (from the **Relays and Coils** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

When the input state from the left becomes on, the value of the associated Bit tag is checked. If the tag is false or 0, then the output state to the right is turned on. If the tag is true or 1, then the output state is kept off.

The relay is checked every time the ladder is **scanned**, so long as the input state remains on.

Configuration Reference

Any Bit tag may be associated with the block: %IX, %MX, %QX, or T_DONE.

C.1.3 Rising Edge Relay (LEC)

When used in a Ladder Diagram, this block acts as a relay that controls the passing of the rung state from the left to the right. The state is passed only when the value of the associated Bit tag changes from 0 to 1.

Select the  tool (from the **Relays and Coils** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

When the value of the associated Bit tag transitions from 0 to 1, the input state from the left is momentarily passed through the the output state to the right.

The state is passed only once, on the first **scan** immediately following the transition. After that, the output state is turned off and kept off until the associated Bit tag transitions again from 0 to 1.

Configuration Reference

Any Bit tag may be associated with the block: %IX, %MX, %QX, or T_DONE.

C.1.4 Falling Edge Relay (TEC)

When used in a Ladder Diagram, this block acts as a relay that controls the passing of the rung state from the left to the right. The state is passed only when the value of the associated Bit tag changes from 1 to 0.

Select the  tool (from the **Relays and Coils** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to [configure](#) it.

Functional Description

When the value of the associated Bit tag transitions from 1 to 0, the input state from the left is momentarily passed through the the output state to the right.

The state is passed only once, on the first [scan](#) immediately following the transition. After that, the output state is turned off and kept off until the associated Bit tag transitions again from 1 to 0.

Configuration Reference

Any Bit tag may be associated with the block: %IX, %MX, %QX, or T_DONE.

C.1.5 Output Coil (OC)

When used in a Ladder Diagram, this block acts as a coil that sets an associated Bit tag to match the state of the rung.

Select the  tool (from the [Relays and Coils](#) toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to [configure](#) it.

Functional Description

This block always passes the input state through to the output state without change; when the input state becomes on, the output state is turned on, and when the input state becomes off, the output state is turned off.

When the input state from the left becomes on, the associated Bit tag is set to 1. When the input state becomes off, the tag is set to 0.

The coil is checked every time the ladder is [scanned](#), regardless of whether the input state is on or off.

Configuration Reference

Any Memory Bit (%MX) or Output Bit (%QX) tag may be associated with the block.

C.1.6 Negated Output Coil (NEGOC)

When used in a Ladder Diagram, this block acts as a coil that sets an associated Bit tag to the inverse of the state of the rung.

Select the  tool (from the **Relays and Coils** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the input state through to the output state without change; when the input state becomes on, the output state is turned on, and when the input state becomes off, the output state is turned off.

When the input state from the left becomes on, the associated Bit tag is set to 0. When the input state becomes off, the tag is set to 1.

The coil is checked every time the ladder is **scanned**, regardless of whether the input state is on or off.

Configuration Reference

Any Memory Bit (%MX) or Output Bit (%QX) tag may be associated with the block.

C.1.7 Latched Coil (LOC)

When used in a Ladder Diagram, this block acts as a coil that sets an associated Bit tag to 1 and maintains it until it is explicitly reset to 0 by an external action.

Select the  tool (from the **Relays and Coils** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the input state through to the output state without change; when the input state becomes on, the output state is turned on, and when the input state becomes off, the output state is turned off.

When the input state from the left becomes on, the associated Bit tag is set to 1. The tag is maintained, or “latched, at 1 until it is explicitly reset to 0 by an external action (either an I/O change or another PointeControl block). The tag is not changed when the input state becomes off.

The coil is checked every time the ladder is **scanned**, so long as the input state remains on. However, the associated Bit tag cannot be set (latched) unless it is already 0.

Configuration Reference

Any Memory Bit (%MX) or Output Bit (%QX) tag may be associated with the block.

C.1.8 Unlatched Coil (UOC)

When used in a Ladder Diagram, this block acts as a coil that sets an associated Bit tag to 0 and maintains it until it is explicitly reset to 1 by an external action.

Select the  tool (from the **Relays and Coils** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the input state through to the output state without change; when the input state becomes on, the output state is turned on, and when the input state becomes off, the output state is turned off.

When the input state from the left becomes off, the associated Bit tag is set to 0. The tag is maintained, or “unlatched, at 0 until it is explicitly reset to 1 by an external action (either an I/O change or another PointeControl block). The tag is not changed when the input state becomes on.

The coil is checked every time the ladder is **scanned**, so long as the input state remains off. However, the associated Bit tag cannot be set (unlatched) unless it is already 1.

Configuration Reference

Any Memory Bit (%MX) or Output Bit (%QX) tag may be associated with the block.

C.1.9 Rising Edge Coil (LEOC)

When used in a Ladder Diagram, this block acts as a coil that momentarily sets an associated Bit tag to 1 when the rung state changes from 0 to 1.

Select the  tool (from the **Relays and Coils** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the input state through to the output state without change; when the input state becomes on, the output state is turned on, and when the input state becomes off, the output state is turned off.

When the input state from the left transitions from 0 to 1, the associated Bit tag is momentarily set to 1. The tag is only maintained for a single execution of the block, after which it is reset to 0.

Configuration Reference

Any Memory Bit (%MX) or Output Bit (%QX) tag may be associated with the block.

C.1.10 Falling Edge Coil (TEOC)

When used in a Ladder Diagram, this block acts as a coil that momentarily sets an associated Bit tag to 1 when the rung state changes from 1 to 0.

Select the  tool (from the **Relays and Coils** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the input state through to the output state without change; when the input state becomes on, the output state is turned on, and when the input state becomes off, the output state is turned off.

When the input state from the left transitions from 1 to 0, the associated Bit tag is momentarily set to 1. The tag is only maintained for a single execution of the block, after which it is reset to 0.

Configuration Reference

Any Memory Bit (%MX) or Output Bit (%QX) tag may be associated with the block.

C.1.11 Falling Edge Detector (F_TRIG)

When used in a Ladder Diagram, the F_TRIG block waits for an input bit to change from 1 to 0 and triggers an output bit when it does.

Select the  tool (from the **Relays and Coils** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the Clock input (**CLK**) is monitored for a transition from 1 to 0. When a transition is detected, the Output (**Q**) is triggered; i.e., **Q** is set to 1 for a single execution of the block, after which it is immediately reset to 0.

The block function is executed every time the ladder is **scanned**, so long as **EN** remains on. However, **Q** is triggered only when a transition in **CLK** is detected. After **Q** has been triggered, **CLK** must be reset to 1 and transition again to 0 before **Q** can be triggered again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.

Param	Name	Config	Var Type	Description
CLK	Clock	req	%IX %MX %QX T_DONE	The input value.
Q	Output	req	%IX %MX %QX	The output value that is triggered when the input value transitions from 1 to 0.

C.1.12 Rising Edge Detector (R_TRIG)

When used in a Ladder Diagram, the R_TRIG block waits for an input bit to change from 0 to 1 and triggers an output bit when it does.

Select the  tool (from the Relays and Coils toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to [configure](#) it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the Clock input (**CLK**) is monitored for a transition from 0 to 1. When a transition is detected, the Output (**Q**) is triggered; i.e., **Q** is set to 1 for a single execution of the block, after which it is immediately reset to 0.

The block function is executed every time the ladder is scanned, so long as **EN** remains on. However, **Q** is triggered only when a transition in **CLK** is detected. After **Q** has been triggered, **CLK** must be reset to 0 and transition again to 1 before **Q** can be triggered again.

Configuration Reference

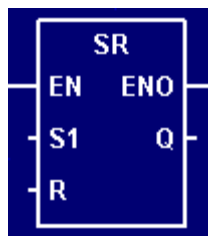
The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
CLK	Clock	req	%IX %MX %QX T_DONE	The input value.
Q	Output	req	%IX %MX %QX	The output value that is triggered when the input value transitions from 0 to 1.

C.1.13 Set-Dominant Bistable (SR)

When used in a Ladder Diagram, the SR block switches an output bit between 0 and 1 depending on the values of two input bits. The block is “set-dominant, meaning that a decision to set the output bit to 1 will override a decision to reset the output bit to 0.

Select the  tool (from the **Relays and Coils** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: a new value for the Output (**Q**) is determined based on the Set and Reset inputs (**S1** and **R**), as well as the existing value of **Q**. If **S1** is true or 1, then **Q** is set to 1. If **R** is true or 1, then **Q** is reset to 0. (However, the block is set-dominant, so **S1** will override **R**.) If both **S1** and **R** are false or 0, then **Q** is left at its existing value regardless of what it is.

Therefore, the block function is evaluated according to the following table:

SET	0	1	0	1
RESET	0	0	1	1
OUTPUT	existing value	1	0	1

The block function is executed every time the ladder is scanned, so long as **EN** remains on.

Configuration Reference

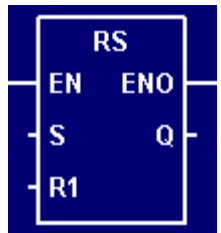
The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
S1	Set	req	%IX %MX %QX T_DONE	The "set" input. This will override the "reset" input.
R	Reset	req	%IX %MX %QX T_DONE Numeric	The "reset" input.
Q	Output	req	%IX %MX %QX	The output value determined by the "set" and "reset" inputs.

C.1.14 Reset-Dominant Bistable (RS)

When used in a Ladder Diagram, the SR block switches an output bit between 0 and 1 depending on the values of two input bits. The block is “reset-dominant, meaning that a decision to reset the output bit to 0 will override a decision to set the output bit to 1.

Select the  tool (from the **Relays and Coils** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: a new value for the Output (**Q**) is determined based on the Set and Reset inputs (**S** and **R1**), as well as the existing value of **Q**. If **S** is true or 1, then **Q** is set to 1. If **R1** is true or 1, then **Q** is reset to 0. (However, the block is reset-dominant, so **R1** will override **S**.) If both **S** and **R1** are false or 0, then **Q** is left at its existing value regardless of what it is.

Therefore, the block function is evaluated according to the following table:

SET	0	1	0	1
RESET	0	0	1	1
OUTPUT	existing value	1	0	0

The block function is executed every time the ladder is **scanned**, so long as **EN** remains on.

Configuration Reference

The parameters of this block are described in the following table:

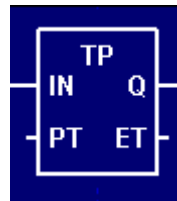
Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
S	Set	req	%IX %MX %QX T_DONE	The "set" input.
R1	Reset	req	%IX %MX %QX T_DONE Numeric	The "reset" input. This will override the "set" input.
Q	Output	req	%IX %MX %QX	The output value determined by the "set" and "reset" inputs.

C.2 Timer and Counter Blocks

C.2.1 Timer, Pulse (TP)

When used in a Ladder Diagram, the TP block turns on the output state for a fixed-width pulse.

Select the  tool (from the **Timers and Counters Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

When the Input State (**IN**) becomes on, the Output State (**Q**) is turned on and the associated **Timer** variable (**TIMER**) is started. When the Timer equals the Preset Time input (**PT**), the Timer is stopped and **Q** is turned off.

When both **IN** and **Q** become off, the Timer is reset to 0.

The Timer cannot be stopped once it is started, even if **IN** becomes off before the Timer equals **PT**.

The Elapsed Time output (**ET**) shows the time passed since **IN** became on; in other words, it shows the current value of the Timer itself. When the Timer reaches **PT**, **ET** remains equal to **PT** until it is reset. When the Timer is reset to 0, **ET** is reset to 0.

Configuration Reference


The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
IN	Input State	no	-	The state of the rung (off/on) received from the left.
Q	Output State	no	-	The state of the rung (off/on) passed to the right.
TIMER	Timer	req	Timer	The Timer variable on which the function is based.

Param	Name	Config	Var Type	Description
PT	Preset Time	opt	%IUD %QUD %MUD Numeric	The length of the pulse, in msec. If no Preset Time is defined, then the preset value of Timer variable is used instead.
ET	Elapsed Time	opt	%QUD %MUD	The time elapsed since the Input State became on, in msec. In most instances, the Elapsed Time equals the current value of the Timer variable.

C.2.2 Timer, ON Delay (TON)

When used in a Ladder Diagram, the TON block turns on the output state after a specified time delay has elapsed.

Select the  tool (from the **Timers and Counters Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

When the Input State (**IN**) becomes on, the associated **Timer** variable (**TIMER**) is started and continues while **IN** remains on. When the Timer equals the Preset Time input (**PT**), the Output State (**Q**) is turned on.

When **IN** becomes off, **Q** is turned off and the Timer is reset to 0.

If **IN** becomes off while the Timer is counting but before it equals **PT**, then the Timer is reset to 0 and **Q** remains off.

The Elapsed Time output (**ET**) shows the time passed since **IN** became on; in other words, it shows the current value of the Timer itself. When the Timer reaches **PT**, **ET** remains equal to **PT** until it is reset. When the Timer is reset to 0, **ET** is reset to 0.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
IN	Input State	no	-	The state of the rung (off/on) received from the left.
Q	Output State	no	-	The state of the rung (off/on) passed to the right.
TIMER	Timer	req	Timer	The Timer variable on which the function is based.
PT	Preset Time	opt	%IUD %QUD %MUD Numeric	The length of the delay, in msec. If no Preset Time is defined, then the preset value of Timer variable is used.
ET	Elapsed Time	opt	%QUD %MUD	The time elapsed since the Input State became on, in msec. In most instances, the Elapsed Time equals the current value of the Timer variable.

C.2.3 Timer, OFF Delay (TOF)

When used in a Ladder Diagram, the TOF block turns off the output state after a specified time delay has elapsed.

Select the  tool (from the **Timers and Counters Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

When the Input State (**IN**) becomes on, the Output State (**Q**) is turned on and the associated **Timer** variable (**TIMER**) is reset to 0.

When **IN** becomes off, the Timer is started and continues while **IN** remains off. When the Timer equals the Preset Time input (**PT**), **Q** is turned off.

If **IN** becomes on while the Timer is counting but before it equals **PT**, then the Timer is reset to 0 and **Q** remains on.

The Elapsed Time output (**ET**) shows the time passed since **IN** became on; in other words, it shows the current value of the Timer itself. When the Timer reaches **PT**, **ET** remains equal to **PT** until it is reset. When the Timer is reset to 0, **ET** is reset to 0.

Configuration Reference

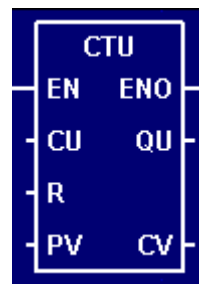
The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
IN	Input State	no	-	The state of the rung (off/on) received from the left.
Q	Output State	no	-	The state of the rung (off/on) passed to the right.
TIMER	Timer	req	Timer	The Timer variable on which the function is based.
PT	Preset Time	opt	%IUD %QUD %MUD Numeric	The length of the delay, in msec. If no Preset Time is defined, then the preset value of Timer variable is used.
ET	Elapsed Time	opt	%QUD %MUD	The time elapsed since the Input State became on, in msec. In most instances, the Elapsed Time equals the current value of the Timer variable.

C.2.4 Counter, Up (CTU)

When used in a Ladder Diagram, the CTU block increments a counter by 1. It also sets a “done bit when the counter reaches a preset value.

Select the  tool (from the **Timers and Counters Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed and the following conditions are evaluated in order:

1. If the Reset input (**R**) is true, then the Counter Value output (**CV**) is reset to 0.
2. If **R** is false and the Count Up input (**CU**) is true, then **CV** is incremented by 1.
3. If **CV** is greater than or equal to the Preset Value input (**PV**), then the Output Up bit (**QU**) is set to true. If **CV** is less than **PV**, then **QU** is set to false.

The block function is executed every time the ladder is scanned, so long as **EN** remains on.

NOTE: Because the frequency of the count is based on the project's **Scan Interval**, it should not be used to gauge real time. To gauge real time, use a **Timer**.

Configuration Reference


The parameters of this block are described in the following table:

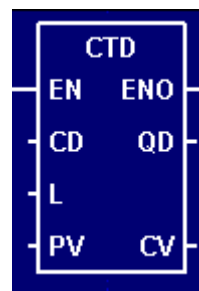
Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
CU	Count Up	req	%IX %QX %MX T_DONE	The enable bit which must be set in order for the counter to increment.
R	Reset	req	%IX %QX %MX T_DONE	The reset bit; if this bit is set, then the counter is set to 0.
PV	Preset Value	req	%IUD %QUD %MUD Numeric	The preset or target value of the counter.
QU	Output Up	req	%QX %MX	The "done" bit which is set when the counter reaches the preset value.

Param	Name	Config	Var Type	Description
CV	Counter Value	req	%QUD %MUD	The current value of the counter.

C.2.5 Counter, Down (CTD)

When used in a Ladder Diagram, the CTD block decrements a counter by 1. It also sets a “done bit when the counter reaches 0.

Select the  tool (from the **Timers and Counters Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed and the following conditions are evaluated in order:

1. If the Preset input (**L**) is true, then the Counter Value output (**CV**) is reset to the Preset Value input (**PV**).
2. If **L** is false and the Count Down input (**CD**) is true, then **CV** is decremented by 1.
3. If **CV** is less than or equal to 0, then the Output Down bit (**QD**) is set to true. If **CV** is greater than 0, then **QD** is set to false.

The block function is executed every time the ladder is **scanned**, so long as **EN** remains on.

NOTE: Because the frequency of the count is based on the project’s **Scan Interval**, it should not be used to gauge real time. To gauge real time, use a **Timer**.


Configuration Reference

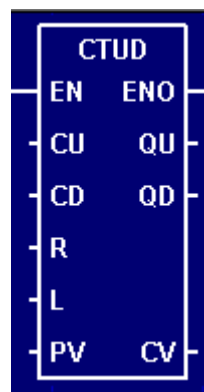
The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
CD	Count Down	req	%IX %QX %MX T_DONE	The enable bit which must be set in order for the counter to decrement.
L	Preset	req	%IX %QX %MX T_DONE	The preset bit; if this bit is set, then the counter is set to the preset value.
PV	Preset Value	req	%IUD %QUD %MUD Numeric	The preset value of the counter.
QD	Output Down	req	%QX %MX	The "done" bit which is set when the counter reaches 0.
CV	Counter Value	req	%QUD %MUD	The current value of the counter.

C.2.6 Counter, Up/Down (CTUD)

When used in a Ladder Diagram, the CTUD block either increments or decrements a counter by 1, depending on which enable bit is set. It also sets "done bit when the counter reaches either a preset value or 0.

Select the  tool (from the **Timers and Counters Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to [configure](#) it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed and the following conditions are evaluated in order:

1. The Reset and Preset inputs (**R** and **L**) are checked and the Counter Value output (**CV**) is changed accordingly:
 - If **R** is true and **L** is false, then **CV** is set to 0 and the function skips to #3 below.
 - If **R** is false and **L** is true, then **CV** is set to the Preset Value input (**PV**) and the function skips to #3 below.
 - If **R** and **L** are both true, then **R** takes precedence (see above).
 - If **R** and **L** are both false, then **CV** is not changed and the function proceeds to #2 below.
2. The Count Up and Count Down inputs (**CU** and **CD**) are checked and **CV** is changed accordingly:
 - If **CU** is true, **CD** is false, and **CV** is less than **PV**, then **CV** is incremented by 1 and the function proceeds to #3 below.
 - If **CU** is false, **CD** is true, and **CV** is greater than 0, then **CV** is decremented by 1 and the function proceeds to #3 below.
 - In all other conditions, **CV** is not changed.
3. The Output Up and Output Down bits (**QU** and **QD**) are set according to the current value of **CV**:
 - If **CV** is greater than or equal to **PV**, then **QU** is set to true and **QD** is set to false.
 - If **CV** is less than or equal to 0, then **QU** is set to false and **QD** is set to true.
 - If **CV** is between 0 and **PV**, then both **QU** and **QD** are set to false.

The block function is executed every time the ladder is [scanned](#), so long as **EN** remains on.

NOTE: Because the frequency of the count is based on the project's [Scan Interval](#), it should not be used to gauge real time. To gauge real time, use a [Timer](#).

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
CU	Count Up	req	%IX %QX %MX T_DONE	The enable bit which must be set in order for the counter to increment.
CD	Count Down	req	%IX %QX %MX T_DONE	The enable bit which must be set in order for the counter to decrement.
R	Reset	req	%IX %QX %MX T_DONE	The reset bit; if this bit is set, then the counter is set to 0.
L	Preset	req	%IX %QX %MX T_DONE	The preset bit; if this bit is set, then the counter is set to the preset value.
PV	Preset Value	req	%IUD %QUD %MUD Numeric	The preset value of the counter.
QU	Output Up	req	%QX %MX	The "done bit which is set when the counter reaches the preset value.
QD	Output Down	req	%QX %MX	The "done bit which is set when the counter reaches 0.
CV	Counter Value	req	%QUD %MUD	The current value of the counter.

C.3 Math Blocks

C.3.1 Add (ADD)

When used in a Ladder Diagram, the ADD block finds the sum of two inputs and sends the result to output.

Select the  tool (from the **Math Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the Input Values 1 and 2 (**IN1** and **IN2**) are added together and the result is placed in the Output Value (**OUT**).

The block function is executed every time the ladder is **scanned**, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.

Param	Name	Config	Var Type	Description
IN1	Input Value 1	req	any Input any Output any Memory T_DONE T_VALUE T_PREVAL Numeric	The first input value.
IN2	Input Value 2	req	any Input any Output any Memory T_DONE T_VALUE T_PREVAL Numeric	The second input value.
OUT	Output Value	req	any Input any Output any Memory	The result of adding the two input values together.

NOTE: This function does not check for bit register overflow. You can assign any variables you wish, but mixing variable sizes – for example, a 32-bit input and an 8-bit output – may result in unusable output.

C.3.2 Subtract (SUB)

When used in a Ladder Diagram, the SUB block subtracts one input from another and sends the result to output.

Select the  tool (from the **Math Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the Input Value 2 (**IN2**) is subtracted from the Input Value 1 (**IN1**) and the result is placed in the Output Value (**OUT**).

The block function is executed every time the ladder is scanned, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

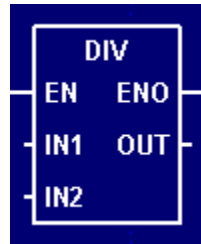
Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN1	Input Value 1	req	any Input any Output any Memory T_DONE T_VALUE T_PREVAL Numeric	The first input value.
IN2	Input Value 2	req	any Input any Output any Memory T_DONE T_VALUE T_PREVAL Numeric	The second input value.
OUT	Output Value	req	any Input any Output any Memory	The result of subtracting the second input value from the first input value.

NOTE: This function does not check for bit register overflow. You can assign any variables you wish, but mixing variable sizes – for example, a 32-bit input and an 8-bit output – may result in unusable output.

C.3.3 Divide (DIV)

When used in a Ladder Diagram, the DIV block divides one input by another and sends the result to output.

Select the  tool (from the **Math Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the Input Value 1 (**IN1**) is divided by the Input Value 2 (**IN2**) and the result is placed in the Output Value (**OUT**).

The block function is executed every time the ladder is **scanned**, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN1	Input Value 1	req	any Input any Output any Memory T_DONE T_VALUE T_PREVAL Numeric	The first input value.

Param	Name	Config	Var Type	Description
IN2	Input Value 2	req	any Input any Output any Memory T_DONE T_VALUE T_PREVAL Numeric	The second input value.
OUT	Output Value	req	any Input any Output any Memory	The result of dividing the first input value by the second input value.

NOTE: This function does not check for bit register overflow. You can assign any variables you wish, but mixing variable sizes – for example, a 32-bit input and an 8-bit output – may result in unusable output.

NOTE: This function cannot perform division by zero. Floating point division by zero generates an overflow condition. Integer division by zero causes an exception and shuts down the application.

C.3.4 Multiply (MUL)

When used in a Ladder Diagram, the MUL block multiplies two inputs and sends the result to output.

Select the  tool (from the **Math Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the Input Value 1 (**IN1**) is multiplied by the Input Value 2 (**IN2**) and the result is placed in the Output Value (**OUT**).

The block function is executed every time the ladder is scanned, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN1	Input Value 1	req	any Input any Output any Memory T_DONE T_VALUE T_PREVAL Numeric	The first input value.
IN2	Input Value 2	req	any Input any Output any Memory T_DONE T_VALUE T_PREVAL Numeric	The second input value.
OUT	Output Value	req	any Input any Output any Memory	The result of multiplying the first input value by the second input value.

NOTE: This function does not check for bit register overflow. You can assign any variables you wish, but mixing variable sizes – for example, a 32-bit input and an 8-bit output – may result in unusable output.

C.3.5 Square Root (SQRT)

When used in a Ladder Diagram, the SQRT block finds the square root of an input and sends the result to output.

Select the  tool (from the **Math Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to [configure](#) it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the square root of the Input Value (**IN**) is calculated and the result is placed in the Output Value (**OUT**).

The block function is executed every time the ladder is [scanned](#), so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN	Input Value	req	real only* Numeric	The input value. If a Numeric value is manually entered, it must be real / floating.
OUT	Output Value	req	real only*	The result of calculating the square root of the input value.

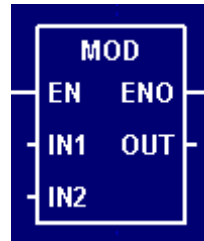
* Any 32-bit Real (F) Input, Output, or Memory tag. For more information, see ["Defining Input, Output, Memory tags"](#) on page 114.

NOTE: This function cannot calculate the square root of a negative input value. If the input value is negative, the value "1.#IND00" (indefinite) is placed in the output. No error is generated.

C.3.6 Modulus (MOD)

When used in a Ladder Diagram, the MOD block finds the remainder from dividing one input value by another input value and sends the result to output.

Select the  tool (from the **Math Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: Input Value 1 (**IN1**) is divided by Input Value 2 (**IN2**) and the remainder is placed in the Output Value (**OUT**).

The block function is executed every time the ladder is **scanned**, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN1	Input Value 1	req	any integer* T_DONE T_VALUE T_PREVAL Numeric	The first input value, which is divided by the second input value.

Param	Name	Config	Var Type	Description
IN2	Input Value 2	req	any integer* T_DONE T_VALUE T_PREVAL Numeric	The second input value, which is divided into the first input value.
OUT	Output Value	req	any integer*	The result of calculating the square root of the input value.

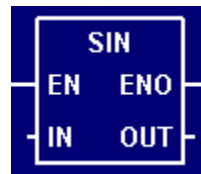
* Any Input, Output, or Memory tag except 32-bit Real (F) or Bit (X). For more information, see "Defining Input, Output, Memory tags" on page 114.

NOTE: This function does not check for bit register overflow. You can assign any variables you wish, but mixing variable sizes – for example, a 32-bit input and an 8-bit output – may result in unusable output.

C.3.7 Sine (SIN)

When used in a Ladder Diagram, the SIN block finds the sine of an input and sends the result to output.

Select the  tool (from the **Math Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the sine of the Input Value (**IN**) is calculated and the result is placed in the Output Value (**OUT**). The input must be specified in radians.

The block function is executed every time the ladder is **scanned**, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

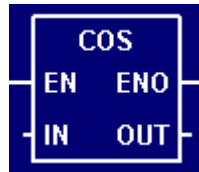
Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN	Input Value	req	real only* Numeric	The input value, in radians. If a Numeric value is entered, it must be real / floating.
OUT	Output Value	req	real only*	The result of calculating the sine of the input value.

* Any 32-bit Real (F) Input, Output, or Memory tag. For more information, see ["Defining Input, Output, Memory tags"](#) on page 114.

C.3.8 Cosine (COS)

When used in a Ladder Diagram, the COS block finds the cosine of an input and sends the result to output.

Select the  tool (from the [Math Blocks](#) toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to [configure](#) it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the cosine of the Input Value (**IN**) is calculated and the result is placed in the Output Value (**OUT**). The input must be specified in radians.

The block function is executed every time the ladder is [scanned](#), so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN	Input Value	req	real only* Numeric	The input value, in radians. If a Numeric value is entered, it must be real / floating.
OUT	Output Value	req	real only*	The result of calculating the cosine of the input value.

* Any 32-bit Real (F) Input, Output, or Memory tag. For more information, see “Defining Input, Output, Memory tags” on page 114.

C.3.9 Tangent (TAN)

When used in a Ladder Diagram, the TAN block finds the tangent of an input and sends the result to output.

Select the  tool (from the **Math Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the tangent of the Input Value (**IN**) is calculated and the result is placed in the Output Value (**OUT**). The input must be specified in radians.

The block function is executed every time the ladder is **scanned**, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN	Input Value	req	real only* Numeric	The input value, in radians. If a Numeric value is entered, it must be real / floating.
OUT	Output Value	req	real only*	The result of calculating the tangent of the input value.

* Any 32-bit Real (F) Input, Output, or Memory tag. For more information, see “Defining Input, Output, Memory tags” on page 114.

C.3.10 Arc Sine (ASIN)

When used in a Ladder Diagram, the ASIN block finds the arc sine of an input and sends the result to output.

Select the  tool (from the **Math Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the arc sine of the Input Value (**IN**) is calculated and the result is placed in the Output Value (**OUT**). The result is expressed in radians.

The block function is executed every time the ladder is **scanned**, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN	Input Value	req	real only* Numeric	The input value. If a Numeric value is entered, it must be real / floating.
OUT	Output Value	req	real only*	The result of calculating the arc sine of the input value, expressed in radians.

* Any 32-bit Real (F) Input, Output, or Memory tag. For more information, see ["Defining Input, Output, Memory tags"](#) on page 114.

C.3.11 Arc Cosine (ACOS)

When used in a Ladder Diagram, the ACOS block finds the arc cosine of an input and sends the result to output.

Select the  tool (from the [Math Blocks](#) toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to [configure](#) it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the arc cosine of the Input Value (**IN**) is calculated and the result is placed in the Output Value (**OUT**). The result is expressed in radians.

The block function is executed every time the ladder is [scanned](#), so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

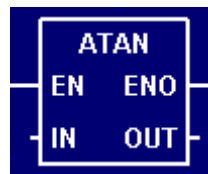
Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN	Input Value	req	real only* Numeric	The input value. If a Numeric value is entered, it must be real / floating.
OUT	Output Value	req	real only*	The result of calculating the arc cosine of the input value, expressed in radians.

* Any 32-bit Real (F) Input, Output, or Memory tag. For more information, see ["Defining Input, Output, Memory tags"](#) on page 114.

C.3.12 Arc Tangent (ATAN)

When used in a Ladder Diagram, the ATAN block finds the arc tangent of an input and sends the result to output.

Select the  tool (from the [Math Blocks](#) toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to [configure](#) it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the arc tangent of the Input Value (**IN**) is calculated and the result is placed in the Output Value (**OUT**). The result is expressed in radians.

The block function is executed every time the ladder is [scanned](#), so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN	Input Value	req	real only* Numeric	The input value. If a Numeric value is entered, it must be real / floating.
OUT	Output Value	req	real only*	The result of calculating the arc tangent of the input value, expressed in radians.

* Any 32-bit Real (F) Input, Output, or Memory tag. For more information, see ["Defining Input, Output, Memory tags"](#) on page 114.

C.3.13 Absolute Value (ABS)

When used in a Ladder Diagram, the ABS block finds the absolute value of an input and sends the result to output.

Select the  tool (from the [Math Blocks](#) toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to [configure](#) it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the absolute value of the Input Value (**IN**) is calculated and the result is placed in the Output Value (**OUT**).

The block function is executed every time the ladder is [scanned](#), so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

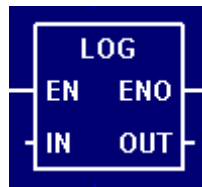
Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN	Input Value	req	signed only* T_DONE T_VALUE T_PREVAL Numeric	The input value.
OUT	Output Value	req	signed only*	The result of calculating the absolute value of the input value.

* Any Input, Output, or Memory tag labeled as "signed. For more information, see "Defining Input, Output, Memory tags" on page 114.

C.3.14 Logarithm (LOG)

When used in a Ladder Diagram, the LOG block finds the base-10 logarithm of an input and sends the result to output.

Select the  tool (from the **Math Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the logarithm of the Input Value (**IN**) is calculated and the result is placed in the Output Value (**OUT**).

The block function is executed every time the ladder is **scanned**, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN	Input Value	req	real only* Numeric	The input value. If a Numeric value is manually entered, it must be real / floating.
OUT	Output Value	req	real only*	The result of calculating the logarithm of the input value.

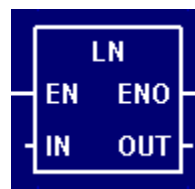
* Any 32-bit Real (F) Input, Output, or Memory tag. For more information, see “Defining Input, Output, Memory tags” on page 114.

NOTE: This function cannot process a zero or negative input value. If the input value is zero, then the value “1.#INF00” (infinite) is placed in the output. If the input value is negative, then the value “- 1.#QNAN0” (not available) is placed in the output. No error is generated.

C.3.15 Natural Logarithm (LN)

When used in a Ladder Diagram, the LN block finds the natural logarithm of an input and sends the result to output.

Select the  tool (from the **Math Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the natural log of the Input Value (**IN**) is calculated and the result is placed in the Output Value (**OUT**).

The block function is executed every time the ladder is scanned, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN	Input Value	req	real only* Numeric	The input value. If a Numeric value is manually entered, it must be real / floating.
OUT	Output Value	req	real only*	The result of calculating the natural log of the input value.

* Any 32-bit Real (F) Input, Output, or Memory tag. For more information, see "Defining Input, Output, Memory tags" on page 114.

NOTE: This function cannot process a zero or negative input value. If the input value is zero, then the value "1.#INF00" (infinite) is placed in the output. If the input value is negative, then the value "- 1.#QNAN0" (not available) is placed in the output. No error is generated.

C.3.16 Exponential (EXPT)

When used in a Ladder Diagram, the EXPT block raises the base input to the power of the exponent input and sends the result to output.

Select the  tool (from the Math Blocks toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to configure it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the Input Value 1 (**IN1**) is raised to the power of the Input Value 2 (**IN2**) and the result is placed in the Output Value (**OUT**).

The block function is executed every time the ladder is scanned, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN1	Input Value 1	req	real only* Numeric	The base input value. If a Numeric value is manually entered, it must be real / floating.
IN2	Input Value 2	req	real only* Numeric	The exponent input value. If a Numeric value is manually entered, it must be real / floating.
OUT	Output Value	req	real only*	The result of raising the base input value to the exponent input value.

* Any 32-bit Real (F) Input, Output, or Memory tag. For more information, see ["Defining Input, Output, Memory tags"](#) on page 114.

NOTE: This function does not check for bit register overflow. If overflow occurs, the value "1.#INF00" (infinite) is placed in the output.

C.3.17 Natural Exponential (EXP)

When used in a Ladder Diagram, the EXP block find the natural exponential of the input and sends the result to output.

Select the  tool (from the **Math Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the natural exponential of the Input Value (**IN**) is calculated and the result is placed in the Output Value (**OUT**).

The block function is executed every time the ladder is **scanned**, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN	Input Value	req	real only* Numeric	The input value. If a Numeric value is manually entered, it must be real / floating.
OUT	Output Value	req	real only*	The result of calculating the natural exponential of the input value.

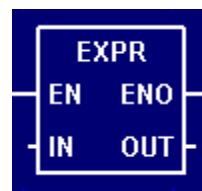
* Any 32-bit Real (F) Input, Output, or Memory tag. For more information, see ["Defining Input, Output, Memory tags"](#) on page 114.

NOTE: This function does not check for bit register overflow. If overflow occurs, the value "1.#INF00" (infinite) is placed in the output.

C.3.18 Expression (EXPR)

As it is currently implemented, the EXPR block simply assigns the value of the input to the output. In future releases this function will allow expressions to be defined and executed as part of the diagram.

Select the  tool (from the **Math Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the Input Value (**IN**) is directly assigned (copied) to the Output Value (**OUT**).

The block function is executed every time the ladder is **scanned**, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.

Param	Name	Config	Var Type	Description
IN	Input Value	req	any Input any Output any Memory T_DONE T_VALUE T_PREVAL Literal Numeric	The input value.
OUT	Output Value	req	any Input any Output any Memory	The result of assigning the input value.

NOTE: This function does not check for bit register overflow. You can assign any variables you wish, but mixing variable sizes – for example, a 32-bit input and an 8-bit output – may result in unusable output.

C.4 Comparison Blocks

C.4.1 Greater Than (GT)

When used in a Ladder Diagram, the GT block checks to see if one input is greater than another and uses the result — false or true — to set the output rung state.

Select the  tool (from the Comparison Blocks toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to [configure](#) it.

Functional Description

When **EN** becomes on, the block function is executed: the Input Value 1 (**IN1**) is compared with the Input Value 2 (**IN2**) and the result is placed in the Output (**Q**).

If **IN1** is greater than **IN2**, then "true is placed in **Q**. If **IN1** is not greater than **IN2**, then "false is placed in **Q**.

The result placed in **Q** determines the state of the rung passed to the right: a "true result turns the rung on and a "false result turns the rung off.

The block function is executed every time the ladder is scanned, so long as **EN** remains on. If **EN** becomes off, then **Q** is immediately turned off regardless of the values of **IN1** and **IN2**.

Configuration Reference

The parameters of this block are described in the following table:

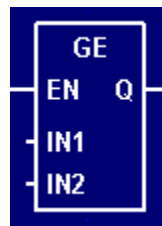
Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
Q	Output	no	-	The result — false or true — of comparing the two input values. The result becomes the state of the rung (off/on) passed to the right.

Param	Name	Config	Var Type	Description
IN1	Input Value 1	req	any Input any Output any Memory T_DONE T_VALUE T_PREVAL	The first input value. The function checks to see if this value is greater than the second input value.
IN2	Input Value 2	req	any Input any Output any Memory T_DONE T_VALUE T_PREVAL Numeric	The second input value.

C.4.2 Greater Than or Equal to (GE)

When used in a Ladder Diagram, the GE block checks to see if one input is greater than or equal to another and uses the result — false or true — to set the output rung state.

Select the  tool (from the Comparison Blocks toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

When **EN** becomes on, the block function is executed: the Input Value 1 (**IN1**) is compared with the Input Value 2 (**IN2**) and the result is placed in the Output (**Q**).

If **IN1** is greater than or equal to **IN2**, then "true is placed in **Q**. If **IN1** is not greater than or equal to **IN2**, then "false is placed in **Q**.

The result placed in **Q** determines the state of the rung passed to the right: a "true result turns the rung on and a "false result turns the rung off.

The block function is executed every time the ladder is scanned, so long as **EN** remains on. If **EN** becomes off, then **Q** is immediately turned off regardless of the values of **IN1** and **IN2**.

Configuration Reference

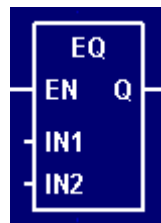
The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
Q	Output	no	-	The result — false or true — of comparing the two input values. The result becomes the state of the rung (off/on) passed to the right.
IN1	Input Value 1	req	any Input any Output any Memory T_DONE T_VALUE T_PREVAL	The first input value. The function checks to see if this value is greater than or equal to the second input value.
IN2	Input Value 2	req	any Input any Output any Memory T_DONE T_VALUE T_PREVAL Numeric	The second input value.

C.4.3 Equal to (EQ)

When used in a Ladder Diagram, the EQ block checks to see if one input is equal to another and uses the result — false or true — to set the output rung state.

Select the  tool (from the Comparison Blocks toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to [configure](#) it.

Functional Description

When **EN** becomes on, the block function is executed: the Input Value 1 (**IN1**) is compared with the Input Value 2 (**IN2**) and the result is placed in the Output (**Q**).

If **IN1** is equal to **IN2**, then “true is placed in **Q**. If **IN1** is not equal to **IN2**, then “false is placed in **Q**.

The result placed in **Q** determines the state of the rung passed to the right: a “true result turns the rung on and a “false result turns the rung off.

The block function is executed every time the ladder is scanned, so long as **EN** remains on. If **EN** becomes off, then **Q** is immediately turned off regardless of the values of **IN1** and **IN2**.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
Q	Output	no	-	The result — false or true — of comparing the two input values. The result becomes the state of the rung (off/on) passed to the right.
IN1	Input Value 1	req	any Input any Output any Memory T_DONE T_VALUE T_PREVAL	The first input value. The function checks to see if this value is equal to the second input value.
IN2	Input Value 2	req	any Input any Output any Memory T_DONE T_VALUE T_PREVAL Numeric	The second input value.

NOTE: This function does not check for bit register overflow. You can assign any variables you wish, but mixing variable sizes – for example, a 32-bit input and an 8-bit output – may result in unusable output.

C.4.4 Not Equal to (NE)

When used in a Ladder Diagram, the NE block checks to see if one input is not equal to another and uses the result — false or true — to set the output rung state.

Select the  tool (from the Comparison Blocks toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to [configure](#) it.

Functional Description

When **EN** becomes on, the block function is executed: the Input Value 1 (**IN1**) is compared with the Input Value 2 (**IN2**) and the result is placed in the Output (**Q**).

If **IN1** is not equal to **IN2**, then "true is placed in **Q**. If **IN1** is not not equal to **IN2**, then "false is placed in **Q**.

The result placed in **Q** determines the state of the rung passed to the right: a "true result turns the rung on and a "false result turns the rung off.

The block function is executed every time the ladder is [scanned](#), so long as **EN** remains on. If **EN** becomes off, then **Q** is immediately turned off regardless of the values of **IN1** and **IN2**.

Configuration Reference

The parameters of this block are described in the following table:

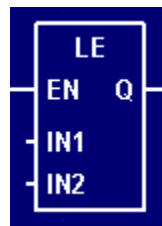
Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
Q	Output	no	-	The result — false or true — of comparing the two input values. The result becomes the state of the rung (off/on) passed to the right.

Param	Name	Config	Var Type	Description
IN1	Input Value 1	req	any Input any Output any Memory T_DONE T_VALUE T_PREVAL	The first input value. The function checks to see if this value is not equal to the second input value.
IN2	Input Value 2	req	any Input any Output any Memory T_DONE T_VALUE T_PREVAL Numeric	The second input value.

C.4.5 Less than or Equal to (LE)

When used in a Ladder Diagram, the LE block checks to see if one input is less than or equal to another and uses the result — false or true — to set the output rung state.

Select the  tool (from the Comparison Blocks toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

When **EN** becomes on, the block function is executed: the Input Value 1 (**IN1**) is compared with the Input Value 2 (**IN2**) and the result is placed in the Output (**Q**).

If **IN1** is less than or equal to **IN2**, then "true" is placed in **Q**. If **IN1** is not less than or equal to **IN2**, then "false" is placed in **Q**.

The result placed in **Q** determines the state of the rung passed to the right: a "true" result turns the rung on and a "false" result turns the rung off.

The block function is executed every time the ladder is scanned, so long as **EN** remains on. If **EN** becomes off, then **Q** is immediately turned off regardless of the values of **IN1** and **IN2**.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
Q	Output	no	-	The result — false or true — of comparing the two input values. The result becomes the state of the rung (off/on) passed to the right.
IN1	Input Value 1	req	any Input any Output any Memory T_DONE T_VALUE T_PREVAL	The first input value. The function checks to see if this value is less than or equal to the second input value.
IN2	Input Value 2	req	any Input any Output any Memory T_DONE T_VALUE T_PREVAL Numeric	The second input value.

C.4.6 Less Than (LT)

When used in a Ladder Diagram, the LT block checks to see if one input is less than another and uses the result — false or true — to set the output rung state.

Select the  tool (from the Comparison Blocks toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to [configure](#) it.

Functional Description

When **EN** becomes on, the block function is executed: the Input Value 1 (**IN1**) is compared with the Input Value 2 (**IN2**) and the result is placed in the Output (**Q**).

If **IN1** is less than **IN2**, then “true is placed in **Q**. If **IN1** is not less than **IN2**, then “false is placed in **Q**.

The result placed in **Q** determines the state of the rung passed to the right: a “true result turns the rung on and a “false result turns the rung off.

The block function is executed every time the ladder is scanned, so long as **EN** remains on. If **EN** becomes off, then **Q** is immediately turned off regardless of the values of **IN1** and **IN2**.

Configuration Reference


The parameters of this block are described in the following table:

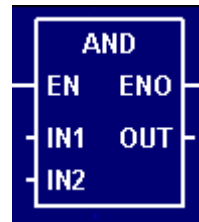
Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
Q	Output	no	-	The result — false or true — of comparing the two input values. The result becomes the state of the rung (off/on) passed to the right.
IN1	Input Value 1	req	any Input any Output any Memory T_DONE T_VALUE T_PREVAL	The first input value. The function checks to see if this value is less than the second input value.
IN2	Input Value 2	req	any Input any Output any Memory T_DONE T_VALUE T_PREVAL Numeric	The second input value.

C.5 Logical and Bit Shift Blocks

C.5.1 And (AND)

When used in a Ladder Diagram, the AND function block performs a bit-for-bit “and comparison between two inputs and sends the result to output.

Select the  tool (from the **Logical and Bit Shift Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: a bit-for-bit logical comparison is made between the Input Values 1 and 2 (**IN1** and **IN2**) and the result is placed in the Output Value (**OUT**).

Each set of bits in **IN1** and **IN2** is evaluated according to the following table:

IN1	0	1	0	1
IN2	0	0	1	1
OUT	0	0	0	1

Therefore, a 16-bit example of AND would be:

```
IN1: 1010010101011101
IN2: 0101011010101110
OUT: 0000010000001100
```

The block function is executed every time the ladder is **scanned**, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN1	Input Value 1	req	any integer* T_DONE T_VALUE T_PREVAL	The first input value.
IN2	Input Value 2	req	any integer* T_DONE T_VALUE T_PREVAL Numeric	The second input value.
OUT	Output Value	req	any integer*	The result of a bit-for-bit AND comparison between the input values.


* Any Input, Output, or Memory tag except 32-bit Real (F). For more information, see ["Defining Input, Output, Memory tags"](#) on page 114.

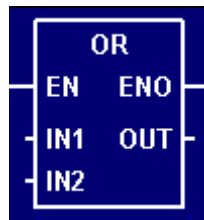
NOTE: No restrictions are placed on the sizes of input or output variables. If the inputs differ in size (for example, an 8-bit byte compared with a 16-bit word), then the smaller input is bit-extended with zeros to match the size of the larger input.

Furthermore, if the result is larger than the output variable (for example, a 16-bit result to be placed in an 8-bit output variable), then the high-order bits of the result are discarded and the low-order bits are placed in the output.

C.5.2 Or (OR)

When used in a Ladder Diagram, the OR block performs a bit-for-bit “or comparison between two inputs and sends the result to output.

Select the  tool (from the **Logical and Bit Shift Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: a bit-for-bit logical comparison is made between the Input Values 1 and 2 (**IN1** and **IN2**) and the result is placed in the Output Value (**OUT**).

Each set of bits in **IN1** and **IN2** is evaluated according to the following table:

IN1	0	1	0	1
IN2	0	0	1	1
OUT	0	1	1	1

Therefore, a 16-bit example of OR would be:

```
IN1: 1010010101011101
IN2: 0101011010101110
OUT: 1111011111111111
```

The block function is executed every time the ladder is **scanned**, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN1	Input Value 1	req	any integer* T_DONE T_VALUE T_PREVAL	The first input value.
IN2	Input Value 2	req	any integer* T_DONE T_VALUE T_PREVAL Numeric	The second input value.
OUT	Output Value	req	any integer*	The result of a bit-for-bit OR comparison between the input values.


* Any Input, Output, or Memory tag except 32-bit Real (F). For more information, see ["Defining Input, Output, Memory tags"](#) on page 114.

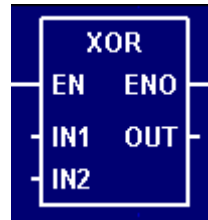
NOTE: No restrictions are placed on the sizes of inputs or output variables. If the inputs differ in size (for example, an 8-bit byte compared with a 16-bit word), then the smaller input is bit-extended with zeros to match the size of the larger input.

Furthermore, if the result is larger than the output variable (for example, a 16-bit result to be placed in an 8-bit output variable), then the high-order bits of the result are discarded and the low-order bits are placed in the output.

C.5.3 Exclusive Or (XOR)

When used in a Ladder Diagram, the XOR block performs a bit-for-bit “exclusive or comparison between two inputs and sends the result to output.

Select the  tool (from the **Logical and Bit Shift Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: a bit-for-bit logical comparison is made between the Input Values 1 and 2 (**IN1** and **IN2**) and the result is placed in the Output Value (**OUT**).

Each set of bits in **IN1** and **IN2** is evaluated according to the following table:

IN1	0	1	0	1
IN2	0	0	1	1
OUT	0	1	1	0

Therefore, a 16-bit example of XOR would be:

```
IN1: 1010010101011101
IN2: 0101011010101110
OUT: 1111001111110011
```

The block function is executed every time the ladder is **scanned**, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN1	Input Value 1	req	any integer* T_DONE T_VALUE T_PREVAL	The first input value.
IN2	Input Value 2	req	any integer* T_DONE T_VALUE T_PREVAL Numeric	The second input value.
OUT	Output Value	req	any integer*	The result of a bit-for-bit XOR comparison between the input values.


* Any Input, Output, or Memory tag except 32-bit Real (F). For more information, see ["Defining Input, Output, Memory tags"](#) on page 114.

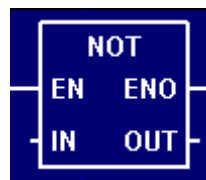
NOTE: No restrictions are placed on the sizes of inputs or output variables. If the inputs differ in size (for example, an 8-bit byte compared with a 16-bit word), then the smaller input is bit-extended with zeros to match the size of the larger input.

Furthermore, if the result is larger than the output variable (for example, a 16-bit result to be placed in an 8-bit output variable), then the high-order bits of the result are discarded and the low-order bits are placed in the output.

C.5.4 Not (NOT)

When used in a Ladder Diagram, the NOT block performs a bit-for-bit inversion upon an input and sends the result to output.

Select the  tool (from the [Logical and Bit Shift Blocks](#) toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to [configure](#) it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: a bit-for-bit logical inversion is made upon the Input Values (**IN**) and the result is placed in the Output Value (**OUT**).

Each bit in **IN** is evaluated according to the following table:

IN	0	1
OUT	1	0

Therefore, a 16-bit example of NOT would be:

```
IN1: 1010010101011101
OUT: 0101101010100010
```

The block function is executed every time the ladder is scanned, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN	Input Value	req	any integer* T_DONE T_VALUE T_PREVAL	The input value.
OUT	Output Value	req	any integer*	The result of a bit-for-bit NOT inversion upon the input value.


* Any Input, Output, or Memory tag except 32-bit Real (F). For more information, see "Defining Input, Output, Memory tags" on page 114.

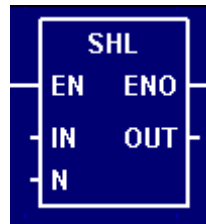
NOTE: If the function result is larger than the output variable (for example, a 16-bit result to be placed in an 8-bit output variable), then the high-order bits of the result are discarded and the low-order bits are placed in the output.

Also, this function does not negate or invert the sign of a signed variable. To invert the sign of a signed variable, configure an XOR function with the variable to be inverted as the first input value and -1 as the second input variable.

C.5.5 Shift bits Left (SHL)

When used in a Ladder Diagram, the SHL block shifts the bits of the input a specified number of places to the left and sends the result to output.

Select the  tool (from the Logical and Bit Shift Blocks toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to [configure](#) it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the Input Value (**IN**) is shifted the specified Number of Places (**N**) to the left and zeroes are placed in the vacated registers. The resulting bit pattern is placed in the Output Value (**OUT**).

Therefore, a 16-bit example of SHL would be:

```
IN: 1010010101011101
N: 5
OUT: 1010101110100000
```

The block function is executed every time the ladder is [scanned](#), so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.

Param	Name	Config	Var Type	Description
IN	Input Value	req	any integer* T_DONE T_VALUE T_PREVAL	The input value.
N	Number of Places	req	unsigned** T_DONE T_VALUE T_PREVAL Numeric	The number of places to be shifted.
OUT	Output Value	req	any integer*	The result of shifting the input value <i>n</i> places to the left.


* Any Input, Output, or Memory tag except 32-bit Real (F) or Bit (X). For more information, see "Defining Input, Output, Memory tags" on page 114.

** Any unsigned (UB, UW, UD, X) Input, Output, or Memory tag. For more information, see "Defining Input, Output, Memory tags" on page 114.

NOTE: If the function result is larger than the output variable (for example, a 16-bit result to be placed in an 8-bit output variable), then the high-order bits of the result are discarded and the low-order bits are placed in the output. No overflow error is generated.

C.5.6 Shift bits Right (SHR)

When used in a Ladder Diagram, the SHR block shifts the bits of the input a specified number of places to the right and sends the result to output.

Select the  tool (from the Logical and Bit Shift Blocks toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to [configure](#) it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the Input Value (**IN**) is shifted the specified Number of Places (**N**) to the right and zeroes are placed in the vacated registers. The resulting bit pattern is placed in the Output Value (**OUT**).

Therefore, a 16-bit example of SHL would be:

```
IN: 1010010101011101
N: 5
OUT: 0000010100101010
```

The block function is executed every time the ladder is scanned, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN	Input Value	req	any integer* T_DONE T_VALUE T_PREVAL	The input value.
N	Number of Places	req	unsigned** T_DONE T_VALUE T_PREVAL Numeric	The number of places to be shifted.
OUT	Output Value	req	any integer*	The result of shifting the input value <i>n</i> places to the right.


* Any Input, Output, or Memory tag except 32-bit Real (F) or Bit (X). For more information, see "Defining Input, Output, Memory tags" on page 114.

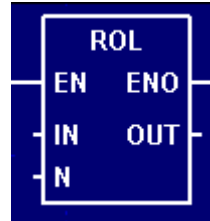
** Any unsigned (UB, UW, UD, X) Input, Output, or Memory tag. For more information, see "Defining Input, Output, Memory tags" on page 114.

NOTE: If the function result is larger than the output variable (for example, a 16-bit result to be placed in an 8-bit output variable), then the high-order bits of the result are discarded and the low-order bits are placed in the output. No overflow error is generated.

C.5.7 Rotate bits Left (ROL)

When used in a Ladder Diagram, the ROL block rotates the bits of the input a specified number of places to the left and sends the result to output.

Select the  tool (from the **Logical and Bit Shift Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the Input Value (**IN**) is rotated the specified Number of Places (**N**) to the left. The bits rotated off the left are added back on the right. The resulting bit pattern is placed in the Output Value (**OUT**).

Therefore, a 16-bit example of ROL would be:

```
IN: 1010010101011101
N: 5
OUT: 1010101110110100
```

The block function is executed every time the ladder is **scanned**, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.

Param	Name	Config	Var Type	Description
IN	Input Value	req	any integer* T_DONE T_VALUE T_PREVAL	The input value.
N	Number of Places	req	unsigned** T_DONE T_VALUE T_PREVAL Numeric	The number of places to be rotated.
OUT	Output Value	req	any integer*	The result of rotating the input value <i>n</i> places to the left.


* Any Input, Output, or Memory tag except 32-bit Real (F) or Bit (X). For more information, see “Defining Input, Output, Memory tags” on page 114.

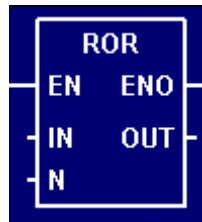
** Any unsigned (UB, UW, UD, X) Input, Output, or Memory tag. For more information, see “Defining Input, Output, Memory tags” on page 114.

NOTE: If the function result is larger than the output variable (for example, a 16-bit result to be placed in an 8-bit output variable), then the high-order bits of the result are discarded and the low-order bits are placed in the output. No overflow error is generated.

C.5.8 Rotate bits Right (ROR)

When used in a Ladder Diagram, the ROR block rotates the bits of the input a specified number of places to the left and sends the result to output.

Select the  tool (from the Logical and Bit Shift Blocks toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to [configure](#) it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the Input Value (**IN**) is rotated the specified Number of Places (**N**) to the right. The bits rotated off the right are added back on the left. The resulting bit pattern is placed in the Output Value (**OUT**).

Therefore, a 16-bit example of ROR would be:

```
IN: 1010010101011101
N: 5
OUT: 1110110100101010
```

The block function is executed every time the ladder is scanned, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN	Input Value	req	any integer* T_DONE T_VALUE T_PREVAL	The input value.
N	Number of Places	req	unsigned** T_DONE T_VALUE T_PREVAL Numeric	The number of places to be rotated.
OUT	Output Value	req	any integer*	The result of rotating the input value <i>n</i> places to the right.

* Any Input, Output, or Memory tag except 32-bit Real (F) or Bit (X). For more information, see "Defining Input, Output, Memory tags" on page 114.

** Any unsigned (UB, UW, UD, X) Input, Output, or Memory tag. For more information, see "Defining Input, Output, Memory tags" on page 114.

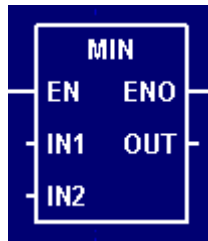
NOTE: If the function result is larger than the output variable (for example, a 16-bit result to be placed in an 8-bit output variable), then the high-order bits of the result are discarded and the low-order bits are placed in the output. No overflow error is generated.

C.6 Selection Blocks

C.6.1 Select minimum value (MIN)

When used in a Ladder Diagram, the MIN block finds the smaller of two inputs and sends the result to output.

Select the  tool (from the Selection Blocks toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to [configure](#) it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the Input Values 1 and 2 (**IN1** and **IN2**) are compared and the smaller is placed in the Output Value (**OUT**).

The block function is executed every time the ladder is [scanned](#), so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.

Param	Name	Config	Var Type	Description
IN1	Input Value 1	req	any Input any Output any Memory T_DONE T_VALUE T_PREVAL Numeric	The first input value.
IN2	Input Value 2	req	any Input any Output any Memory T_DONE T_VALUE T_PREVAL Numeric	The second input value.
OUT	Output Value	req	any Input any Output any Memory	The result of finding the smaller of the two input values.

NOTE: This function does not check for bit register overflow. You can assign any variables you wish, but mixing variable sizes – for example, a 32-bit input and an 8-bit output – may result in unusable output.

C.6.2 Select maximum value (MAX)

When used in a Ladder Diagram, the MAX block finds the larger of two inputs and sends the result to output.

Select the  tool (from the Selection Blocks toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to [configure](#) it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the Input Values 1 and 2 (**IN1** and **IN2**) are compared and the larger is placed in the Output Value (**OUT**).

The block function is executed every time the ladder is **scanned**, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN1	Input Value 1	req	any Input any Output any Memory T_DONE T_VALUE T_PREVAL Numeric	The first input value.
IN2	Input Value 2	req	any Input any Output any Memory T_DONE T_VALUE T_PREVAL Numeric	The second input value.
OUT	Output Value	req	any Input any Output any Memory	The result of finding the larger of the two input values.

NOTE: This function does not check for bit register overflow. You can assign any variables you wish, but mixing variable sizes – for example, a 32-bit input and an 8-bit output – may result in unusable output.

C.6.3 Limit value (LIM)

When used in a Ladder Diagram, the LIM block limits an input to a specified range.

Select the  tool (from the Selection Blocks toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to [configure](#) it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the Input Value (**IN**) is compared to both the Minimum and Maximum Values (**MIN** and **MAX**) and the limit is placed in the Output Value (**OUT**). When **IN** is less than **MIN**, **MIN** is placed in **OUT**. When **IN** is greater than **MAX**, **MAX** is placed in **OUT**. Otherwise, **IN** is placed directly in **OUT**.

The block function is executed every time the ladder is scanned, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.

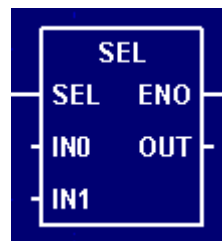
Param	Name	Config	Var Type	Description
IN	Input Value	req	any Input any Output any Memory T_DONE T_VALUE T_PREVAL Numeric	The input value.
MIN	Minimum Value	req	any Input any Output any Memory T_DONE T_VALUE T_PREVAL Numeric	The minimum value against which the input value is compared.
MAX	Maximum Value	req	any Input any Output any Memory T_DONE T_VALUE T_PREVAL Numeric	The maximum value against which the input value is compared.
OUT	Output Value	req	any Input any Output any Memory	The result of finding the limit.

NOTE: This function does not check for bit register overflow. You can assign any variables you wish, but mixing variable sizes – for example, a 32-bit input and an 8-bit output – may result in unusable output.

C.6.4 Select one of two values (SEL)

When used in a Ladder Diagram, the SEL block selects one of two input values, depending on the rung state received from the left, and sends the result to output.

Select the  tool (from the Selection Blocks toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Select input state (**SEL**) through to the Enable Out output state (**ENO**) without change; when **SEL** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

The block function is executed every time the ladder is scanned, regardless of whether **SEL** is on or off. The state of **SEL** determines which Input Value (**IN0** or **IN1**) is selected and placed in the Output Value (**OUT**). If **SEL** is on, then **IN1** is placed in **OUT**. If **SEL** is off, then **IN0** is placed in **OUT**.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
SEL	Select	no	-	The state of the rung (off/on) received from the left. Also determines which input value is selected to place in the output.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN0	Input Value OFF	req	any Input any Output any Memory T_DONE T_VALUE T_PREVAL Numeric	The input value that is selected if the rung is off.
IN1	Input Value ON	req	any Input any Output any Memory T_DONE T_VALUE T_PREVAL Numeric	The input value that is selected if the rung is on.
OUT	Output Value	req	any Input any Output any Memory	The result of selecting one of two input values.

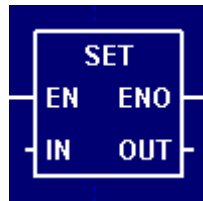
NOTE: This function does not check for bit register overflow. You can assign any variables you wish, but mixing variable sizes – for example, a 32-bit input and an 8-bit output – may result in unusable output.

C.7 String Blocks

C.7.1 Set string (SET)

When used in a Ladder Diagram, the SET block copies a string (variable or literal) from an input to an output.

Select the  tool (from the **String Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the Input String (**IN**) is copied directly to the Output String (**OUT**).

The block function is executed every time the ladder is **scanned**, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

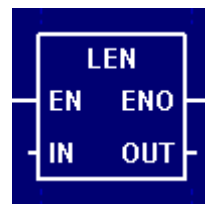
Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN	Input String	req	String Literal	The input string. Literals must be enclosed in single quotes (' ').
OUT	Output String	req	String	The result of copying the input string. The output is always NULL-terminated.

NOTE: If the defined Element Length of the String variable configured to **OUT** is smaller than the result of the function, then the result is truncated to fit. No overflow error is generated.

C.7.2 Find string length (LEN)

When used in a Ladder Diagram, the LEN block finds the character length of a string (variable or literal) and send it to output.

Select the  tool (from the **String Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the length (i.e., the number of characters) of the Input String (**IN**) is determined and the result is placed in the Output Value (**OUT**).

The block function is executed every time the ladder is **scanned**, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

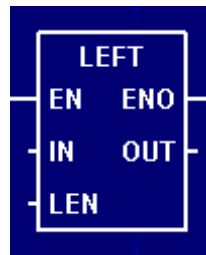
Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN	Input String	req	String Literal	The input string. Literals must be enclosed in single quotes (' ').
OUT	Output Value	req	any integer*	The result of finding the length the input string.

* Any Input, Output, or Memory tag except 32-bit Real (F) or Bit (X). For more information, see "Defining Input, Output, Memory tags" on page 114.

C.7.3 Extract sub-string from left (LEFT)

When used in a Ladder Diagram, the LEFT block extracts a sub-string of specified length from the left end of a string (variable or literal) and sends the result to an output.

Select the  tool (from the [String Blocks](#) toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to [configure](#) it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: a sub-string of up to specified Length (**LEN**) is extracted from the Input String (**IN**) and placed in the Output String (**OUT**). The sub-string is extracted from the left end of **IN**.

For example:

```
IN: 'Hello world. This is PointeControl.'  
LEN: 8  
OUT: 'Hello wo'
```

The block function is executed every time the ladder is [scanned](#), so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN	Input String	req	String Literal	The input string. Literals must be enclosed in single quotes (' ').
LEN	Length	req	unsigned* Numeric	The length (i.e., the number of characters) of the sub-string to be extracted.
OUT	Output String	req	String	The result of extracting the sub-string. The output is always NULL-terminated.

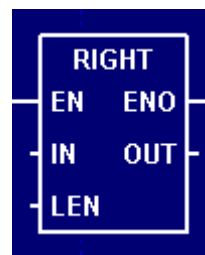
* Any unsigned (UB, UW, UD) Input, Output, or Memory tag. For more information, see ["Defining Input, Output, Memory tags"](#) on page 114.

NOTE: If the configured length of the string variable assigned to **OUT** is smaller than the result of the function, then the result is truncated to fit. No overflow error is generated.

C.7.4 Extract sub-string from right (RIGHT)

When used in a Ladder Diagram, the RIGHT block extracts a sub-string of specified length from the right end of a string (variable or literal) and sends the result to an output.

Select the  tool (from the [String Blocks](#) toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to [configure](#) it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: a sub-string of up to specified Length (**LEN**) is extracted from the Input String (**IN**) and placed in the Output String (**OUT**). The sub-string is extracted from the right end of **IN**.

For example:

```
IN: 'Hello world. This is PointeControl.'
LEN: 8
OUT: 'Control.'
```

The block function is executed every time the ladder is **scanned**, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN	Input String	req	String Literal	The input string. Literals must be enclosed in single quotes (' ').
LEN	Length	req	unsigned* Numeric	The length (i.e., the number of characters) of the sub-string to be extracted.
OUT	Output String	req	String	The result of extracting the sub-string. The output is always NULL-terminated.

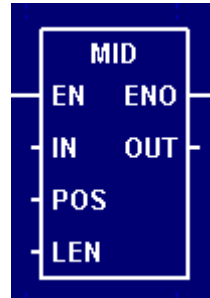
* Any unsigned (UB, UW, UD) Input, Output, or Memory tag. For more information, see ["Defining Input, Output, Memory tags"](#) on page 114.

NOTE: If the configured length of the string variable assigned to **OUT** is smaller than the result of the function, then the result is truncated to fit. No overflow error is generated.

C.7.5 Extract sub-string from middle (MID)

When used in a Ladder Diagram, the MID block extracts a sub-string of specified length from the middle of a string (variable or literal) and sends the result to an output.

Select the  tool (from the **String Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: a sub-string of up to specified Length (**LEN**) is extracted from the Input String (**IN**), starting at the specified Position (**POS**). The result is placed in the Output String (**OUT**).

For example:

```
IN: 'Hello world. This is PointeControl.'
POS: 9
LEN: 8
OUT: 'rld. Thi'
```

The block function is executed every time the ladder is **scanned**, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

NOTE: If there are fewer than **POS** characters in the Input String, then the output will be a null string.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN	Input String	req	String Literal	The input string. Literals must be enclosed in single quotes (' ').
POS	Position	req	unsigned* Numeric	The starting position of the sub-string to be extracted. The first character corresponds to a Position of 1.
LEN	Length	req	unsigned* Numeric	The length (i.e., the number of characters) of the sub-string to be extracted.
OUT	Output String	req	String	The result of extracting the sub-string. The output is always NULL-terminated.

* Any unsigned (UB, UW, UD) Input, Output, or Memory tag. For more information, see ["Defining Input, Output, Memory tags"](#) on page 114.

NOTE: If the configured length of the string variable assigned to **OUT** is smaller than the result of the function, then the result is truncated to fit. No overflow error is generated.

C.7.6 Concatenate strings (CAT)

When used in a Ladder Diagram, the CAT block concatenates two strings (variable or literal) and send the resulting string to an output.

Select the  tool (from the [String Blocks](#) toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to [configure](#) it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the Input Strings 1 and 2 (**IN1** and **IN2**) are concatenated and the resulting string is placed in the Output String (**OUT**).

For example:

```
IN1: 'Hello world.'
IN2: ' This is PointeControl.'
OUT: 'Hello world. This is PointeControl.'
```

NOTE: This function does not insert any spaces between the concatenated strings.

The block function is executed every time the ladder is scanned, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

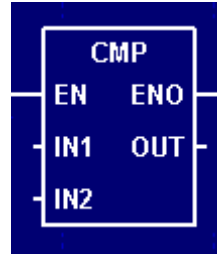
Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN1	Input String 1	req	String Literal	The first input string. Literals must be enclosed in single quotes (' ').
IN2	Input String 2	req	String Literal	The second input string. Literals must be enclosed in single quotes (' ').
OUT	Output String	req	String	The result of concatenating the input strings. The output is always NULL-terminated.

NOTE: If the configured length of the string variable assigned to **OUT** is smaller than the result of the function, then the result is truncated to fit. No overflow error is generated.

C.7.7 Compare strings (CMP)

When used in a Ladder Diagram, the CMP block compares two strings (variable or literal) and send the result to an output.

Select the  tool (from the **String Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the Input Strings 1 and 2 (**IN1** and **IN2**) are compared and the result is placed in the Output Value (**OUT**).

Values for **OUT** are determined as follows:

When...	OUT is...
IN1 < IN2	-1
IN1 = IN2	0
IN1 > IN2	1

When two strings are identical up to the NULL terminator in the shorter string ('ain1' and 'ain100' for example) the shorter string is considered less than the longer.

The block function is executed every time the ladder is **scanned**, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

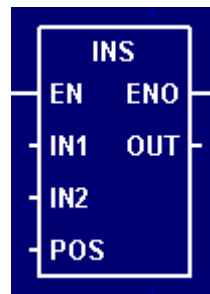
Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN1	Input String 1	req	String Literal	The first input string. Literals must be enclosed in single quotes (' ').
IN2	Input String 2	req	String Literal	The second input string. Literals must be enclosed in single quotes (' ').
OUT	Output Value	req	signed*	The result of comparing the input strings.

* Any signed (B, W, D) Input, Output, or Memory tag. For more information, see "Defining Input, Output, Memory tags" on page 114.

C.7.8 Insert sub-string (INS)

When used in a Ladder Diagram, the INS block inserts a sub-string into an input string (variable or literal) at a specified position and sends the resulting string to an output.

Select the  tool (from the **String Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the Input String 2 (**IN2**) is inserted into the Input String 1 (**IN1**) at the specified Position (**POS**). The resulting string is placed in the Output String (**OUT**).

For example:

```
IN1: 'Hello world.'
IN2: 'This is PointeControl.'
POS: 9
OUT: 'Hello woThis is PointeControl.rld.'
```

NOTE: If there are fewer than **POS** characters in **IN1**, then **IN1** and **IN2** are simply concatenated.

The block function is executed every time the ladder is scanned, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN1	Input String 1	req	String Literal	The input string into which the sub-string is inserted. Literals must be enclosed in single quotes (' ').
IN2	Input String 2	req	String Literal	The sub-string to be inserted. Literals must be enclosed in single quotes (' ').
POS	Position	req	unsigned* Numeric	The starting position where the sub-string will be inserted. The first character corresponds to a Position of 1.
OUT	Output String	req	String	The result of inserting the sub-string. The output is always NULL-terminated.

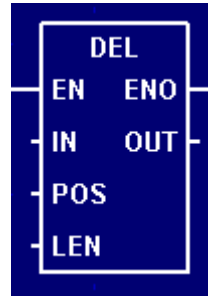
* Any unsigned (UB, UW, UD) Input, Output, or Memory tag. For more information, see "Defining Input, Output, Memory tags" on page 114.

NOTE: If the configured length of the string variable assigned to **OUT** is smaller than the result of the function, then the result is truncated to fit. No overflow error is generated.

C.7.9 Delete sub-string (DEL)

When used in a Ladder Diagram, the DEL block deletes a sub-string of specified length from the middle of a string (variable or literal) and sends the resulting string to an output.

Select the  tool (from the **String Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: a sub-string of specified Length (**LEN**) is deleted from the Input String (**IN**), starting at the specified Position (**POS**). The resulting string is placed in the Output String (**OUT**).

For example:

```
IN: 'Hello world. This is PointeControl.'
POS: 9
LEN: 8
OUT: 'Hello was is PointeControl.'
```

NOTE: If there are fewer than **POS** characters in **IN**, then **IN** is copied without changes to **OUT**.

The block function is executed every time the ladder is **scanned**, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN	Input String	req	String Literal	The input string. Literals must be enclosed in single quotes (' ').
POS	Position	req	unsigned* Numeric	The starting position of the sub-string to be extracted. The first character corresponds to a Position of 1.
LEN	Length	req	unsigned* Numeric	The length (i.e., the number of characters) of the sub-string to be deleted.
OUT	Output String	req	String	The result of deleting the sub-string. The output is always NULL-terminated.

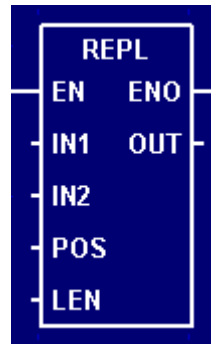
* Any unsigned (UB, UW, UD) Input, Output, or Memory tag. For more information, see ["Defining Input, Output, Memory tags"](#) on page 114.

NOTE: If the configured length of the string variable assigned to **OUT** is smaller than the result of the function, then the result is truncated to fit. No overflow error is generated.

C.7.10 Replace sub-string (REPL)

When used in a Ladder Diagram, the REPL block replaces part of an input string with a specified number of characters from another input string, starting at a specified position. The resulting string is sent to output.

Select the  tool (from the **String Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: a sub-string in Input String 1 (**IN1**), starting at the specified Position (**POS**) and of specified Length (**LEN**), is replaced with Input String 2 (**IN2**). The resulting string is placed in the Output String (**OUT**).

For example:

```
IN1: 'Hello world.'
IN2: 'GOODBYE'
POS: 4
LEN: 5
OUT: 'HelGOODBYErld.'
```

NOTE: If there are fewer than **POS** characters in **IN1**, then **IN1** and **IN2** are simply concatenated.

The block function is executed every time the ladder is **scanned**, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN1	Input String 1	req	String Literal	The input string into which the sub-string is inserted. Literals must be enclosed in single quotes (' ').
IN2	Input String 2	req	String Literal	The sub-string to be inserted. Literals must be enclosed in single quotes (' ').
POS	Position	req	unsigned* Numeric	The starting position where the sub-string will be replaced. The first character corresponds to a Position of 1.
LEN	Length	req	unsigned* Numeric	The length (i.e., the number of characters) of the sub-string to be replaced.
OUT	Output String	req	String	The result of replacing the sub-string. The output is always NULL-terminated.

* Any unsigned (UB, UW, UD) Input, Output, or Memory tag. For more information, see ["Defining Input, Output, Memory tags"](#) on page 114.

NOTE: If the configured length of the string variable assigned to **OUT** is smaller than the result of the function, then the result is truncated to fit. No overflow error is generated.

C.7.11 Find sub-string (FIND)

When used in a Ladder Diagram, the FIND block finds the first occurrence, if any, of a sub-string within a given string. The position of the sub-string is sent to output.

Select the  tool (from the **String Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to [configure](#) it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the Input String 1 (**IN1**) is searched for the first occurrence, if any, of the Input String 2 (**IN2**). If **IN2** is found within **IN1**, then the starting position is placed in the Output Value (**OUT**). If **IN2** is not found, then 0 is placed in **OUT**.

For example:

```
IN1: 'This is OpenControl.'  
IN2: 'is'  
OUT: 3
```

The block function is executed every time the ladder is [scanned](#), so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN1	Input String 1	req	String Literal	The input string to be searched. Literals must be enclosed in single quotes.
IN2	Input String 2	req	String Literal	The sub-string to be found in the input string. Literals must be enclosed in single quotes.
OUT	Output Value	req	any integer*	The resulting position of the sub-string, if found.

* Any Input, Output, or Memory tag except 32-bit Real (F) or Bit (X). For more information, see ["Defining Input, Output, Memory tags"](#) on page 114.

C.8 Flow Control Blocks

C.8.1 Call sub-ladder diagram (CALL)

When used in a Ladder Diagram, this block executes a specified Sub-Ladder.

Select the  tool (from the **Flow Control Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to configure it.

NOTE: This block is configured differently than other Ladder Diagram blocks. See “Configuration Reference” below.

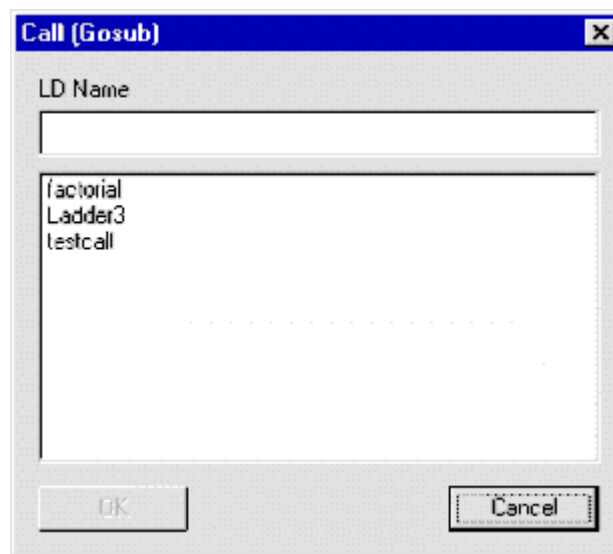
Functional Description

This block always passes the input state through to the output state without change; when the input state becomes on, the output state is turned on, and when the input state becomes off, the output state is turned off.

When the input state from the left becomes on, the Sub-Ladder referenced by the block is executed. Ladder diagram execution continues on the right of the CALL object only after execution of the Sub-Ladder is completed.

Configuration Reference

This block may reference any Ladder Diagram which has been **defined as a Sub-Ladder**. The Sub-Ladder is selected using the dialog below:



All valid Sub-Ladders in the current project will be listed in the dialog. A selection may be made from the list or entered directly into the **LD Name** field.

The **OK** button will be enabled when a valid selection has been made or entered. Click it to save your selection and close the dialog.

C.8.2 Return to main diagram (RETN)

When used in a Sub-Ladder, this block stops execution of the Sub-Ladder and returns to the Ladder Diagram which **called** it. Use of this block is required only if the logic flow of your program requires an early return from the Sub-Ladder (i.e., the Sub-Ladder must be aborted before its normal end). Use of the block is optional in the last rung of a Sub-Ladder.

Select the  tool (from the **Flow Control Blocks** toolbar) and click on a ladder rung to insert the following block:



This block cannot be configured.

NOTE: This block can only be used in a **properly defined Sub-Ladder**. If a Return block is inserted in a regular Ladder Diagram, it will be detected and reported when the diagram's **integrity is checked**.

Functional Description

This block does not pass the rung state in any situation. When the input state from the left becomes on, execution of the Sub-Ladder is immediately stopped and returned to the Ladder Diagram which originally called the Sub-Ladder.

Configuration Reference

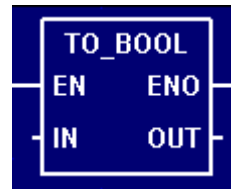
There are no configurable parameters for this block.

C.9 Miscellaneous Blocks

C.9.1 Convert to Boolean (TO_BOOL)

When used in a Ladder Diagram, the TO_BOOL block converts any input value into an equivalent boolean (bit) tag. All non-zero inputs are converted to 1, while zero inputs are converted to 0.

Select the  tool (from the **Miscellaneous Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the Input Value (**IN**) is converted into its equivalent boolean (bit) value and the result is sent to the Output Value (**OUT**). If **IN** is any non-zero value, then **OUT** is set to 1. If **IN** is zero, then **OUT** is set to 0.

NOTE: When **IN** is a String, conversion to a numeric representation is attempted. If the conversion succeeds, then **OUT** is set to 1. If the conversion fails, then **OUT** is set to 0.

The block function is executed every time the ladder is **scanned**, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.

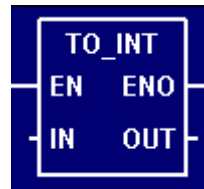
Param	Name	Config	Var Type	Description
IN	Input Value	req	any Input any Memory any Output Numeric String	The input value.
OUT	Output Value	req	%MX %QX	The result of converting the input value into an equivalent boolean (bit) tag.

C.9.2 Convert to Integer (TO_INT)

When used in a Ladder Diagram, the TO_INT block converts any input value into an equivalent integer. It is used primarily to convert floating-point variable types into integer variable types of the same approximate value. However, any input values can be given.

NOTE: Floating point numbers are rounded when converted in this way.

Select the  tool (from the **Miscellaneous Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the Input Value (**IN**) is converted into an equivalent integer and the result is sent to the Output Value (**OUT**).

NOTE: When **IN** is a String, conversion to a numeric representation is attempted. If the conversion succeeds, then **OUT** is set to 1. If the conversion fails, then **OUT** is set to 0.

The block function is executed every time the ladder is **scanned**, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN	Input Value	req	any Input any Memory any Output Numeric String	The input value.
OUT	Output Value	req	any integer*	The result of converting the input value into an equivalent integer.

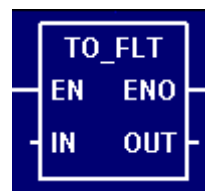
* Any Output or Memory tag except 32-bit Real (F) or Bit (X). For more information, see ["Defining Input, Output, Memory tags"](#) on page 114.

NOTE: This function does not check for bit register overflow. You can assign any variables you wish, but mixing variable sizes – for example, a 32-bit input and an 8-bit output – may result in unusable output.

C.9.3 Convert to Float (TO_FLT)

When used in a Ladder Diagram, the TO_FLT block converts any input value into an equivalent floating point number. It is used primarily to convert integer variable types into floating-point variable types of the same value. However, any input values can be given.

Select the  tool (from the [Miscellaneous Blocks](#) toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to [configure](#) it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the Input Value (**IN**) is converted into an equivalent floating point and the result is sent to the Output Value (**OUT**).

NOTE: When **IN** is a String, conversion to a numeric representation is attempted. If the conversion succeeds, then **OUT** is set to 1. If the conversion fails, then **OUT** is set to 0.

The block function is executed every time the ladder is scanned, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN	Input Value	req	any Input any Memory any Output Numeric String	The input value.
OUT	Output Value	req	%MF %QF	The result of converting the input value into an equivalent floating point.

C.9.4 Convert to String (TO_STRG)

When used in a Ladder Diagram, the TO_STRG block converts any input value into an equivalent string. It is used primarily to convert non-String variables into String variables which can then be stored, edited, or displayed. However, any input values can be given, including other Strings.

Select the  tool (from the Miscellaneous Blocks toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the Input Value (**IN**) is converted into an equivalent string and the result is sent to the Output Value (**OUT**).

For example, the numeric value 123 is converted into a three-character string '123'.

The block function is executed every time the ladder is scanned, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

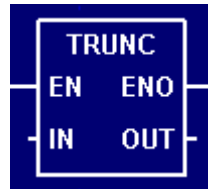
Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN	Input Value	req	any Input any Memory any Output Numeric String	The input value.
OUT	Output Value	req	String	The result of converting the input value into an equivalent string. The output is always NULL-terminated.

NOTE: No restrictions are placed on the size of the output string. If the string resulting from a non-string **IN** is longer than the defined Element Length of the String variable configured to **OUT**, then a NULL string is placed in **OUT**. String-to-string "conversions will simply truncate excess characters. In neither case will an overflow error be generated.

C.9.5 Truncate (TRUNC)

When used in a Ladder Diagram, the TRUNC block truncates a floating-point input value and discards its fractional (decimal) part. This is effectively the same as rounding the value down to the nearest integer, although the value always remains a floating-point number.

Select the  tool (from the **Miscellaneous Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the Input Value (**IN**) is truncated and its fractional (decimal) part is discarded. The result is sent to the Output Value (**OUT**).

For example, the floating-point value 123.45 is truncated to 123.00.

The block function is executed every time the ladder is **scanned**, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN	Input Value	req	%IF %MF %QF	The input value.
OUT	Output Value	req	%MF %QF	The result of truncating the input value.

C.9.6 Integer to Character (TO_CHR)

When used in a Ladder Diagram, the TO_CHR block converts a decimal value into the equivalent ASCII character.

Select the  tool (from the **Miscellaneous Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the Input Value (**IN**) is converted into an equivalent ASCII character and the result is placed in the first character position of the Output Value (**OUT**).

For example, the decimal value 65 is converted into the ASCII character 'A'.

The block function is executed every time the ladder is **scanned**, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN	Input Value	req	%IUB %MUB %QUB	The input value.
OUT	Output Value	req	String	The result of converting the input value into an equivalent ASCII character.

NOTE: The output string is not NULL-terminated, as this is not a string operation.

C.9.7 Character to Integer (CHR_TO)

When used in a Ladder Diagram, the CHR_TO block converts the first character of a string into the equivalent ASCII decimal value.

Select the  tool (from the **Miscellaneous Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the first character of the Input Value (**IN**) is converted into an equivalent ASCII decimal value and the result is placed in the Output Value (**OUT**).

For example, the character 'A' is converted into the ASCII decimal value 65.

The block function is executed every time the ladder is **scanned**, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

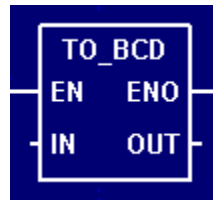
Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN	Input Value	req	String Literal	The input value. Literals must be enclosed in single quotes (' ').
OUT	Output Value	req	any integer*	The result of converting the first character of the input value into an equivalent decimal value.

* Any Memory or Output tag except 32-bit Real (F) or Bit (X). For more information, see ["Defining Input, Output, Memory tags"](#) on page 114.

C.9.8 Integer to BCD (TO_BCD)

When used in a Ladder Diagram, the TO_BCD block converts a regular integer value into an equivalent binary coded decimal (BCD).

Select the  tool (from the Miscellaneous Blocks toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to [configure](#) it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: each digit of the Input Value (**IN**) is separately converted into a 4-bit “nibble”, and then the nibbles are concatenated into a single binary which is placed in the Output Value (**OUT**).

For example, an integer value of 5319 would be converted in the following manner:

DIGIT	5	3	1	9
NIBBLE	0101	0011	0001	1001

The resulting BCD is 0101001100011001.

NOTE: Given the 32-bit limit on the size of Logic Memory variables, the largest integer that can be practically converted into a BCD is 99999999.

The block function is executed every time the ladder is [scanned](#), so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN	Input Value	req	unsigned*	The input value.
OUT	Output Value	req	any integer**	The result of converting the input value into a binary coded decimal (BCD). NOTE: Although the variable type is an integer, the value stored in the variable is still a BCD.

* Any unsigned (UB, UW, UD) Input or Memory tag. For more information, see “Defining Input, Output, Memory tags” on page 114.

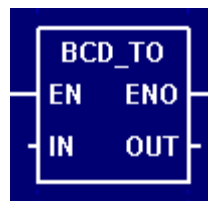
** Any Memory or Output tag except 32-bit Real (F) or Bit (X). For more information, see “Defining Input, Output, Memory tags” on page 114.

NOTE: This function does not check for bit register overflow. You can assign any variables you wish, but if the size of the resulting BCD exceeds the available space in the output variable, then the high-order nibbles are discarded.

C.9.9 BCD to Integer (BCD_TO)

When used in a Ladder Diagram, the BCD_TO block converts a binary coded decimal (BCD) into an equivalent integer value.

Select the  tool (from the Miscellaneous Blocks toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to [configure](#) it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: each 4-bit “nibble of the Input Value (**IN**) is separately converted into a base-10 digit, and then the digits are concatenated into a single integer which is placed in the Output Value (**OUT**).

For example, a BCD of 0101001100011001 would be converted in the following manner:

NIBBLE	0101	0011	0001	1001
DIGIT	5	3	1	9

The resulting integer is 5319.

NOTE: Checks are not performed on the magnitude of the decimal digits in the BCD input value. Values within each nibble are multiplied by the appropriate power of 10 whether they exceed 9 or not. No “out-of-range indication is made when invalid BCD digits are present.

The block function is executed every time the ladder is scanned, so long as **EN** remains on. If **EN** becomes off, then **OUT** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
IN	Input Value	req	any integer*	The input value, read as a BCD.
OUT	Output Value	req	any integer**	The result of converting the input value into an integer.

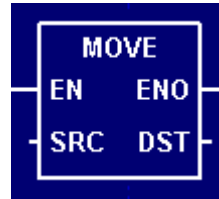
* Any Input or Memory tag except 32-bit Real (F) or Bit (X). For more information, see “Defining Input, Output, Memory tags” on page 114.

** Any Memory or Output tag except 32-bit Real (F) or Bit (X). For more information, see “Defining Input, Output, Memory tags” on page 114.

C.9.10 Move (MOVE)

When used in a Ladder Diagram, the MOVE block directly copies the value from a source variable to a destination variable.

Select the  tool (from the **Miscellaneous Blocks** toolbar) and click on a ladder rung to insert the following block:



Once the block is inserted, you can double-click on it to **configure** it.

Functional Description

This block always passes the Enable input state (**EN**) through to the Enable Out output state (**ENO**) without change; when **EN** becomes on, **ENO** is turned on, and when **EN** becomes off, **ENO** is turned off.

When **EN** becomes on, the block function is executed: the current value in the Source input (**SRC**) is moved (copied) to the Destination output (**DST**). No other manipulation is performed on the value.

The block function is executed every time the ladder is **scanned**, so long as **EN** remains on. If **EN** becomes off, then **DST** remains at its last calculated value until **EN** becomes on and the block function is executed again.

Configuration Reference

The parameters of this block are described in the following table:

Param	Name	Config	Var Type	Description
EN	Enable	no	-	The state of the rung (off/on) received from the left.
ENO	Enable Out	no	-	The state of the rung (off/on) passed to the right.
SRC	Source	req	any Input any Memory any Output Numeric	The input value.
DST	Desti- nation	req	any Input any Memory any Output	The result of moving (copying) the input value.

NOTE: This function does not check for bit register overflow. You can assign any variables you wish, but mixing variable sizes – for example, a 32-bit input and an 8-bit output – may result in unusable output.
