

# PREFACE

Thank you for purchasing the LT integrated development software, "LT Editor Ver. 2.0", hereafter referred to as the "LT Editor".

Please read this manual carefully in order to use this software properly, and be sure to keep this manual handy for future reference.

## NOTES

- (1) The copyrights to all programs and manuals included in the LT Editor Ver. 2.0 Operation Manual - Screen Creation Guide (hereinafter referred to as "this product") are reserved by the Digital Electronics Corporation. Digital grants the use of this product to its users as described in the "Software Operating License Conditions" documentation, included with this product's CD-ROM. Any actions violating the above-mentioned conditions are prohibited by both Japanese and foreign regulations.
- (2) The contents of this manual have been thoroughly inspected. However, if you should find any errors or omissions in this manual, please inform your local LT Editor representative of your findings.
- (3) Regardless of article (2), the Digital Electronics Corporation shall not be held responsible for any damages or third party claims resulting from the use of this product.
- (4) Differences may occur between the descriptions found in this manual and the actual functioning of this product. Therefore, the latest information on this product is provided in data files (i.e. Readme.txt files, etc. ) and in separate documents. Please consult these sources as well as this manual prior to using the product.
- (5) Even though the information contained in and displayed by this product may be related to intangible or intellectual properties of the Digital Electronics Corporation or third parties, the Digital Electronics Corporation shall not warrant or grant the use of said properties to any users and/or other third parties. Digital Electronics Corporation accepts no liability for issues related to the intellectual property rights of third parties or any issues related to the use of the information contained in or displayed by this product.
- (6) The specifications set out in this manual are for overseas products only. As a result, some differences may exist between the specifications given here and for those of the identical Japanese product.

© Copyright 2002 Digital Electronics Corporation. All rights reserved.  
Digital Electronics Corporation, November 2002

The LogiTouch is referred to as "LT" in this manual.

For the rights to trademarks and trade names, see "TRADEMARK RIGHTS".

# TABLE OF CONTENTS

<b>PREFACE .....</b>	<b>1</b>
<b>TABLE OF CONTENTS .....</b>	<b>2</b>
<b>TRADEMARK RIGHTS .....</b>	<b>8</b>
<b>MANUAL SYMBOLS AND TERMINOLOGY .....</b>	<b>9</b>
<b>LT SERIES .....</b>	<b>10</b>
<b>HOW TO USE THIS MANUAL .....</b>	<b>11</b>
<b>PRECAUTIONS .....</b>	<b>13</b>
<b>SETUP GUIDE (Tutorial) .....</b>	<b>17</b>

## 1 CREATING A PROGRAM

<b>1.1 How to Start the LT Editor .....</b>	<b>1-7</b>
<b>1.2 Creating Variables .....</b>	<b>1-8</b>
1.2.1 Creating a Variable List .....	1-8
1.2.2 Selecting Variable Types .....	1-9
1.2.3 Saving Your Program .....	1-10
<b>1.3 Inserting Rungs, Instructions, and Branches .....</b>	<b>1-11</b>
1.3.1 Inserting a Rung .....	1-11
1.3.2 Deleting a Rung .....	1-12
1.3.3 Inserting Instructions .....	1-13
1.3.4 Deleting Instructions .....	1-16
1.3.5 Copying and Pasting Instructions .....	1-17
1.3.6 Inserting Branches .....	1-18
1.3.7 Initialization Logic .....	1-19
<b>1.4 Assigning Variables to Instructions .....</b>	<b>1-21</b>
1.4.1 Instruction Parameter Box .....	1-21
1.4.2 Entering Variables .....	1-22
1.4.3 Completing the Program .....	1-24
<b>1.5 Documenting a Ladder Logic Program .....</b>	<b>1-27</b>
1.5.1 Adding a Program Description .....	1-27
1.5.2 Adding a Rung Description .....	1-28
1.5.3 Adding Descriptions to Variables .....	1-29
1.5.4 Description List Dialog Box .....	1-30

<b>1.6</b>	<b>Copying, Cutting and Pasting Rungs .....</b>	<b>1–31</b>
1.6.1	Copying a Rung .....	1–31
1.6.2	Pasting a Rung .....	1–31
1.6.3	Cut Command .....	1–32
<b>1.7</b>	<b>Subroutines and Labels .....</b>	<b>1–33</b>
1.7.1	Inserting a Subroutine .....	1–33
1.7.2	Inserting Labels .....	1–35
<b>1.8</b>	<b>Navigating a Ladder Logic Program .....</b>	<b>1–36</b>
1.8.1	The [Find] Command .....	1–36
1.8.2	The [References] Command .....	1–37
1.8.3	[References] Dialog Box with Other Dialog Boxes .....	1–39
1.8.4	Using Bookmarks .....	1–40
1.8.5	Using the [Go To Rung] Command .....	1–41
1.8.6	Using the [Go To Label] Command .....	1–41
<b>1.9</b>	<b>I/O Configuration .....</b>	<b>1–42</b>
1.9.1	Assigning Variables to I/O .....	1–42
1.9.2	Unassigning Variables from the [Configure I/O] Dialog Box .....	1–49
1.9.3	Assigning I/O to Variables .....	1–50
1.9.4	Converting I/O Configuration Data .....	1–50
<b>1.10</b>	<b>Checking the Validity of a Program .....</b>	<b>1–52</b>
<b>1.11</b>	<b>Printing Your Ladder Logic Program .....</b>	<b>1–54</b>
<b>1.12</b>	<b>Importing/Exporting a Logic Program .....</b>	<b>1–56</b>
1.12.1	Export .....	1–56
1.12.2	Import .....	1–58
<b>1.13</b>	<b>Developing a Screen Program .....</b>	<b>1–61</b>

## **2 RUNNING THE LADDER LOGIC PROGRAM**

<b>2.1</b>	<b>Configuring the LT Controller .....</b>	<b>2–1</b>
2.1.1	Writing to the Controller .....	2–4
2.1.2	Going to Monitoring Mode .....	2–5
<b>2.2</b>	<b>Starting and Stopping the Controller .....</b>	<b>2–6</b>
<b>2.3</b>	<b>Troubleshooting Using System Variables .....</b>	<b>2–8</b>
<b>2.4</b>	<b>Viewing System Variables .....</b>	<b>2–9</b>
<b>2.5</b>	<b>Reading from the Controller .....</b>	<b>2–10</b>
<b>2.6</b>	<b>Property .....</b>	<b>2–10</b>

**3 ON-LINE EDITING**

3.1 Before Editing ..... 3-1  
3.2 Using Colors for On-Line Editing ..... 3-2  
3.3 Turning a Discrete ON and OFF ..... 3-3  
3.4 Forcing Discrete ON and OFF ..... 3-4  
3.5 Changing Variable Values ..... 3-5  
3.6 Changing Variable Attributes ..... 3-6  
3.7 Data Watch List ..... 3-7

**4 ERRORS AND WARNINGS**

**5 GLOSSARY OF TERMS**

**6 CONTROLLER FEATURES**

6.1 Operating the LT ..... 6-1  
6.1.1 Controller Feature Overview ..... 6-2  
6.1.2 RUN Mode ..... 6-4  
6.1.3 LT Scan Overview ..... 6-5

**7 VARIABLES**

7.1 Variable Names ..... 7-1  
7.2 Variable Types ..... 7-4  
7.3 Accessing Variables ..... 7-7

**8 SYSTEM VARIABLES**

8.1 System Variable List ..... 8-1  
8.1.1 How to Use System Variables ..... 8-2  
8.2 System Variable Details ..... 8-3  
8.2.1 #AvgLogicTime ..... 8-3  
8.2.2 #AvgScanTime ..... 8-3  
8.2.3 #Clock100ms ..... 8-4  
8.2.4 #Day ..... 8-5  
8.2.5 #ForceCount ..... 8-5  
8.2.6 #IOStatus ..... 8-6  
8.2.7 #LogicTime ..... 8-6  
8.2.8 #Month ..... 8-7  
8.2.9 #Platform ..... 8-7  
8.2.10 #ScanCount ..... 8-7

8.2.11 #ScanTime .....	8-8
8.2.12 #Status .....	8-8
8.2.13 #Time .....	8-9
8.2.14 #Version .....	8-10
8.2.15 #Year .....	8-10
8.2.16 #Weekday .....	8-10
8.2.17 #FaultCode .....	8-11
8.2.18 #FaultRung .....	8-12
8.2.19 #IOFault .....	8-12
8.2.20 #Overflow .....	8-13
8.2.21 #Command .....	8-14
8.2.22 #DisableAutoStart .....	8-14
8.2.23 #Fault .....	8-14
8.2.24 #FaultOnMinor .....	8-15
8.2.25 #PercentAlloc .....	8-15
8.2.26 #Screen .....	8-15
8.2.27 #TargetScan .....	8-16
8.2.28 #WatchdogTime .....	8-16

## **9 INSTRUCTIONS**

<b>9.1 Instruction List .....</b>	<b>9-1</b>
<b>9.2 Instruction Details .....</b>	<b>9-5</b>
9.2.1 NO (Normally Open) .....	9-5
9.2.2 NC (Normally Closed) .....	9-6
9.2.3 OUT/M (Output Coil) .....	9-7
9.2.4 NEG (Negated Coil) .....	9-8
9.2.5 SET (Set Coil) .....	9-9
9.2.6 RST (Reset Coil) .....	9-10
9.2.7 PT (Positive Transition Contact) .....	9-11
9.2.8 NT (Negative Transition Contact) .....	9-12
9.2.9 AND (And) .....	9-13
9.2.10 OR (Or) .....	9-14
9.2.11 XOR (Exclusive OR) .....	9-15
9.2.12 NOT (Bit Invert) .....	9-16
9.2.13 MOV (Transfer) .....	9-16
9.2.14 BMOV (Block Transfer) .....	9-18
9.2.15 FMOV (Fill Transfer) .....	9-19

9.2.16 ROL (Rotate Left) .....	9–20
9.2.17 ROR (Rotate Right) .....	9–21
9.2.18 SHL (Shift Left) .....	9–22
9.2.19 SHR (Shift Right) .....	9–23
9.2.20 ADD (Add) .....	9–25
9.2.21 SUB (Subtract) .....	9–26
9.2.22 MUL (Multiply) .....	9–27
9.2.23 DIV (Divide) .....	9–28
9.2.24 MOD (Modulus) .....	9–29
9.2.25 INC (Increment) .....	9–29
9.2.26 DEC (Decrement) .....	9–30
9.2.27 EQ (Compare: =) .....	9–31
9.2.28 GT (Compare: >) .....	9–31
9.2.29 LT (Compare: <) .....	9–32
9.2.30 GE (Compare: >=) .....	9–33
9.2.31 LE (Compare: <=) .....	9–33
9.2.32 NE (Compare: <>) .....	9–34
9.2.33 PID (PID Calculation) .....	9–36
9.2.34 TON (Timer ON Delay) .....	9–49
9.2.35 TOF (Timer OFF Delay) .....	9–51
9.2.36 TP (Timer Pulse) .....	9–53
9.2.37 CTU (UP Counter) .....	9–55
9.2.38 CTD (DOWN Counter) .....	9–56
9.2.39 CTUD (UP/DOWN Counter) .....	9–57
9.2.40 BCD (BCD Conversion) .....	9–58
9.2.41 BIN (Binary Conversion) .....	9–59
9.2.42 ENCO (Encode) .....	9–60
9.2.43 DECO (Decode) .....	9–61
9.2.44 JMP (Jump) .....	9–62
9.2.45 JSR (Jump Subroutine) .....	9–63
9.2.46 RET (Return Subroutine) .....	9–63
9.2.47 FOR/NEXT (Repeat) .....	9–64

## **10 LS AREA REFRESH**

<b>10.1 LS Area Refresh Overview .....</b>	<b>10–1</b>
<b>10.2 LS Area Refresh Settings .....</b>	<b>10–2</b>
10.2.1 LS Area When not using a Device/PLC .....	10–3
<b>10.3 LT and External Device Data Sharing .....</b>	<b>10–6</b>
10.3.1 LS Area Refresh Cautions .....	10–8

## 11 I/O DRIVERS

<b>11.1 I/O Driver Overview</b> .....	<b>11-1</b>
<b>11.2 Flex Network I/F Driver</b> .....	<b>11-2</b>
11.2.1 Flex Network I/F Unit Self-Diagnosis .....	11-2
11.2.2 Communication Check .....	11-3
11.2.3 Error S-No. ....	11-4
11.2.4 I/O Monitor (I/O Connection Check) .....	11-4
11.2.5 Flex Network I/F Unit Troubleshooting .....	11-10
<b>11.3 DIO Driver</b> .....	<b>11-12</b>
11.3.1 DIO Unit Self-Diagnosis.....	11-12
11.3.2 I/O Monitor (I/O Connection Check) .....	11-14
11.3.3 DIO Unit Troubleshooting .....	11-15

## 12 ERROR MESSAGES

<b>12.1 Error Message List</b> .....	<b>12-1</b>
<b>12.2 Error Codes</b> .....	<b>12-3</b>
<b>12.3 Program Errors</b> .....	<b>12-4</b>

# TRADEMARK RIGHTS

The company names and product names used in this manual are the trade names, trademarks (including registered trademarks), and service marks of their respective companies.

This product omits individual descriptions of each of these rights.

<b>Trademark / Trade Name</b>	<b>Right Holder</b>
Microsoft, MS, MS-DOS, Windows, Windows 95, Windows 98, Windows Me, Windows NT, Windows 2000, Windows XP, Windows Explorer, Microsoft Excel 95	Microsoft Corporation, USA
Intel, Pentium	Intel Corporation, USA
Pro-face, Flex Network	Digital Electronics Corporation (in Japan and other countries)
Ethernet	Western Digital Electric Corporation, USA
IBM, VGA, PC/AT	International Business Machines Corporation (IBM), USA

The following terms used in this manual differ from the above mentioned formal trade names and trademarks.

<b>Terms Used in this Manual</b>	<b>Formal Tradename or Trademark</b>
Windows 95	Microsoft® Windows95® Operating System
Windows 98	Microsoft® Windows98® Operating System
Windows Me	Microsoft® WindowsMe® Operating System
Windows NT	Microsoft® WindowsNT® Operating System
Windows 2000	Microsoft® Windows2000® Operating System
Windows XP	Microsoft® WindowsXP® Operating System
MS-DOS	Microsoft® MS-DOS® Operating System








# MANUAL SYMBOLS AND TERMINOLOGY

This manual uses the following symbols and terminology.





## ■ Safety Symbols and Terms

This manual uses the following symbols and terms to identify important information related to the correct and safe operation of this product.

Symbol	Description
	Indicates a potentially hazardous situation that could result in serious injury or death.
	Indicates a potentially hazardous situation that could result in minor injury or equipment damage.
	Indicates a potentially damaging action or dangerous situation that could result in abnormal equipment operation or data loss.
	Indicates instructions or procedures that must be performed to ensure correct product use.
	Indicates instructions or procedures that must not be performed.

## ■ General Information Symbols and Terms










This manual uses the following symbols and terms for general information.

Symbol	Description
	Provides hints on correct product use, or supplementary information.
	Indicates an item's related information (manual name, chapter, section, sub-section).
	Refers to keys on the computer keyboard.  ■ <b>Keyboard Compatibility List</b>
Device	Indicates peripheral devices such as temperature controllers, inverters, etc. connected via serial I/O. It does not include devices connected via the Flex Network or DIO.
LT	Generic name for the "LT Series" Graphic Logic Controller made by Digital Electronics Corporation.
LT Editor	Indicates Digital Electronics Corporation's LT integrated development software "LT Editor" Version 2.0 (this product).

■ **Keyboard Compatibility List**

This manual uses the following symbols to indicate computer keyboard keys.

The key names used by your computer keyboard may differ. Please use the chart below for reference.

Symbol \ Type	PS/2 Compatible 101 Keyboard
	Esc
	Tab 
	Ctrl
	 Shift
	Alt
	Delete
	Backspace

# LT SERIES

The LT Editor supports the following LT models.

Series	Type	Product	Model
<b>LT Series</b>	Type A1	GLC150B-XY32SK	GLC150-BG41-XY32SK-24V
	Type A1	GLC150B-XY32SC	GLC150-BG41-XY32SC-24V
	Type B	GLC150B-RSFL	GLC150-BG41-FLEX-24V
	Type B+	GLC150B-XY32KF	GLC150-BG41-XY32KF-24V
	Type C	GLC150B-XY32SK	GLC150-BG41-RSFL-24V
	Type H1	GLC150B-ADK	GLC150-BG41-ADK-24V
		GLC150B-ADPK	GLC150-BG41-ADPK-24V
		GLC150B-ADTK	GLC150-BG41-ADTK-24V
	Type H2	GLC150B-ADC	GLC150-BG41-ADC-24V
		GLC150B-ADPC	GLC150-BG41-ADPC-24V
GLC150B-ADTC		GLC150-BG41-ADTC-24V	



**Note:** For the types of Device/PLCs supported by the LT Editor, please refer to the "Device Connection Manual".

**Reference** *HOW TO USE THIS MANUAL*

# HOW TO USE THIS MANUAL

## ■ Structure of the Manual

The *LT Editor Ver. 2.0 Operation Manual – Logic Programming Guide* is the first in a series of manuals for this product and explains how to use the LT Editor. There are three other manuals in the series as well as online help.

**Reference** Refer to the *Operation Manual – Screen Creation Guide, Chapter 1 – “LT Editor Fundamentals”* and *1.6 – “LT Editor Manuals and Help”* for an overview of this product.

In addition to these manuals, data files containing supplemental information on updated functions are also provided. To read these additional data files, click on the [Start] button in your Windows OS main screen and select the [Programs] → [Proface] → [LT] menu. Then, click on the [Read Me] selection.

For detailed information on LT series products, please refer to "LT Series User Manual". (Optionally available)

Included in CD-ROM	<b>Operation Manual Screen Creation Guide</b>	Describes the operating procedures for the LT Editor and all functions except for Logic Program development (provided as a PDF file).
	<b>Operation Manual Logic Programming Guide (this manual)</b>	Describes logic program development. The manual consists of three sections: "Installation," "Programming," and "Features." In the Installation Section, you can learn the basic procedures to create a logic program. The Programming Section explains how to operate the LT Editor through a tutorial lesson while the Features Section explains the software settings required for the combination of the LT main unit and the LT Editor. This manual is provided as a PDF file.
	<b>Parts List</b>	Describes the LT Editor's pre-made Parts and symbols (provided as PDF data).
	<b>Device Connection Manual</b>	Describes the methods for connecting the LT to devices of various manufacturers (provided as a PDF file).
Available in the LT Editor screen	<b>Online Help</b>	Describes the methods for setting the LT Editor's windows and dialog boxes, instructions, and functions of logic programs as well as how to set each driver.



**Note:**

- Address settings described in these manuals are for explanatory purposes only. Appropriate addresses must be set according to your requirements.

**Reference** Refer to the *Device Connection Manual*.



- **If you have any questions about the contents of this manual, please contact your local LT distributor. LT distributors will answer to your technical inquiries and provide you with technical consultation.**

**Reference** *Refer to the Screen Creation Guide, Appendix 4 – “Software Trouble Report.”*

- **If you have any question about your personal computer or Microsoft® Windows®, please contact your PC distributor or manufacturer.**

## ■ Chapter Breakdown

This manual consists of three sections: “Installation”, “Programming” and “Features”.

The following is a general description of each chapter.

### [Setup Section]

The Setup Section uses an example application program to describe the basic steps involved in creating a logic program with the LT Editor software. For detailed information about LT Editor operating procedures, or variables/instructions, please refer to the Programming or Features sections.

### [Programming Section]

The Programming Section uses a tutorial to explain how to operate the Logic Program Editor software. In this section you will create a complete ladder logic program.

## ◆ CHAPTER 1: CREATING A PROGRAM

This chapter’s tutorial explains how to create a ladder logic program.

## ◆ CHAPTER 2: RUNNING THE LADDER LOGIC PROGRAM

This chapter explains how to transfer a completed ladder logic program to the LT and then run the program.

## ◆ CHAPTER 3: ON-LINE EDITING

This chapter describes how to use On-Line mode to confirm the execution of the ladder logic program.

## ◆ CHAPTER 4: ERRORS AND WARNINGS

This chapter describes error messages that may be displayed when checking errors with the Logic Program Editor.

## ◆ CHAPTER 5: GLOSSARY OF TERMS

This chapter explains many of the terms used in the Logic Program Editor.

### [Features Section]

The Features section describes how the LT unit operates, and provides a list of instructions and variables used in the ladder logic program.

## ◆ CHAPTER 6: CONTROLLER FEATURES

This chapter describes the operation of the LT unit’s controller.

### ◆ CHAPTER 7: VARIABLES

This chapter provides definitions of the variables used in the ladder logic program and how to use them.

### ◆ CHAPTER 8: SYSTEM VARIABLES

This chapter lists variables that are predefined by the controller.

### ◆ CHAPTER 9: INSTRUCTIONS

This chapter lists instructions that the Logic Program Editor supports.

### ◆ CHAPTER 10: LS AREA REFRESH

This chapter describes how to use the LS area, which is used for control, as well as for sharing data with display features and external devices.

### ◆ CHAPTER 11: I/O DRIVERS

This chapter describes each I/O driver available. It also explains self-diagnostic and troubleshooting procedures.

### ◆ CHAPTER 12: ERROR MESSAGES

This chapter describes the error messages that may appear during LT Editor operation.



**Note:** Online Help also provides detailed Logic Program Editor information.

## PRECAUTIONS

### ■ CD-ROM Usage Precautions

To prevent CD-ROM damage, please observe the following instructions:



- Do not turn your PC ON and OFF with the CD-ROM in the drive.



- Do not remove the CD-ROM from the CD-ROM drive while the drive's operation lamp is lit.
- Do not touch the CD-ROM recording surface.
- Do not place CD-ROMs in a place where they may be exposed to extremely high or low temperatures, high humidity, or dust.

### ■ Product Usage Precautions

To prevent a program malfunction or accident, be sure to observe the following instructions:



**Touch panel switches should NOT be used as a device's emergency stop switch. Generally speaking, all industrial machinery/systems including robots must be equipped with an emergency stop switch that only operates manually. Also, for other machinery/systems, similar manual switches must be provided to ensure safe operation.**



- Do not turn off your personal computer's power switch during the execution of a program.
- Do not change the contents of this product's project files using a text editor software.

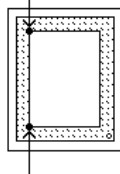
### ■ Restrictions

The LT Editor has the following restrictions.

#### ◆ Drawing

- The LT Editor's display screen uses your PC's character fonts and graphic functions. As a result, there may cause some differences in the appearance between the LT and PC after the screen is transferred to the LT.
- When an LT unit is vertically installed, the panel's coordinates will differ from those used on the screen editor software. Therefore, when you enter screen coordinates using Parts or D-Script, be sure to consider the LT's orientation.

**(0, 0) on the screen editor software**



**(0, 0) on the LT series' panel**

#### ◆ Functions and Settings

- Certain functions and settings supported by the LT unit are not supported by the LT Editor, and vice versa.

##### **[Settings and functions supported by the LT unit (Not by LT Editor) ]**

- Language Font selection
- LT Date/Time settings
- LT Self-Diagnostics Function

##### **[Functions and settings supported by the LT Editor (Not by LT unit)]**

The following settings are included in the "LT System Settings" area:

- "Checksum Verification" settings
- Screen Change Order in hierarchical display mode
- Screen Change according to standby mode time
- Shift to OFFLINE mode settings
- Setting the frequency of Keypad Display processing performed per scan
- LT unit's internal memory (LS area) backup function settings
- "Error Display Reset" settings
- "Watch Dog Timer" settings
- Communication Monitoring Period settings (Designate transmission wait time)

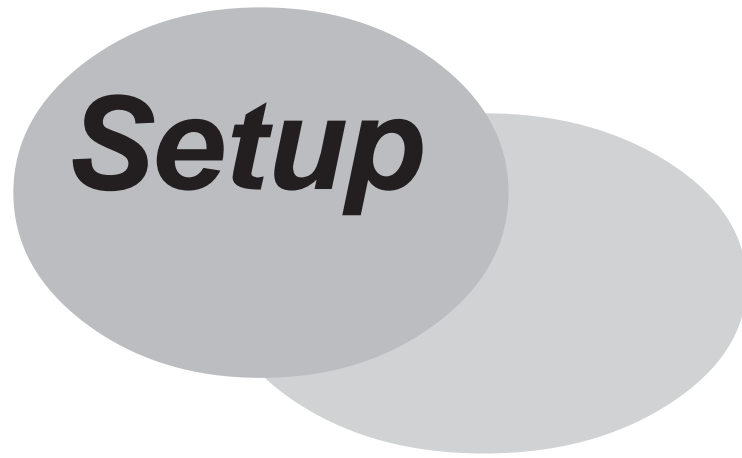
**◆ Logic Program Restrictions**

- LT variables are handled in 32-bit device Low/High order.
- Parts used for display function cannot handle real numbers.
- Values different from the input values may appear during monitoring due to the difference in the real number accuracy between a personal computer and the LT.
- If the LT's logic time (scan time) becomes too long, the sampling time designated for the trend graph may not be accurately maintained.
- When using the Memory Link Method, a change in the value of a variables may not be completely displayed by a trend graph.
- All LT Retentive Variable data is saved using a lithium battery in SRAM backup memory. The battery can back up data for approximately 60 days in its initial condition (fully charged), and for approximately 6 days when the battery's life is almost finished. If you need to back up data for a longer period, you will need to either back up data to your host computer, or configure your system so that data is backed up by LT Editor.
- When a logic program and screen data use the same LS area, be sure to designate all logic symbol LS variables (LS<\*>).

**◆ For users of previous versions**

Version 1.04 and previous version users are required to note the following:

- Do not download or monitor Version 1.04 or earlier version logic programs on an LT that has been set up using LT Editor Version 2.0..





# SETUP GUIDE (Tutorial)

This chapter provides step-by-step instructions for using the LT Editor software to create LT application programs. For a detailed description of how to use the Logic Program Editor and the Drawing Board, please refer to either “Programming” in this manual, the Operation Manual’s “Screen Creation Guide” section, or Online Help.

## ■ Before Starting the Tutorial

The lessons in this chapter describe the procedures for developing application programs with the LT Editor, and explains the basic functions and operations through this step-by-step tutorial. Pro-face recommends that first-time LT Editor users go through all of the tutorial prior to developing application programs.

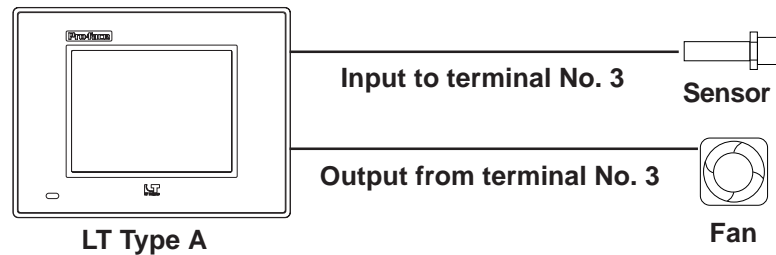
This section describes the procedures for creating a sample application using the following devices.

Also, the LT Editor software must be installed before starting logic programming.

## ■ Equipment List

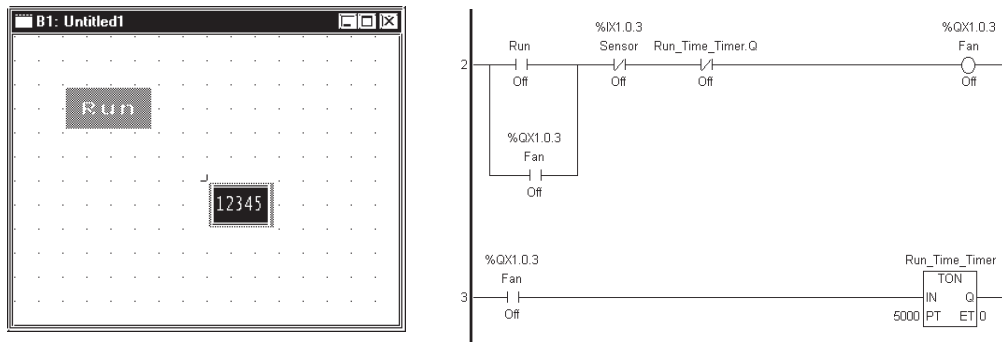
Main Unit	LT Type A
I/O Unit	Built-in Type-A (Combining 16-point input/16-point output)
Cable	Screen transfer cable
Fan	24 VDC fan
Sensor	24 VDC proximity switch

## ■ System Diagram



### ■ Example Application

This example will create the following screen and logic program.



### ■ Explanation

- The fan rotates for five seconds after the LT screen's switch is touched.
- The fan's stop time can be changed using the keypad displayed on the LT screen.
- The fan will stop rotating if a signal is received from the sensor.

### ■ Developing an Application Program

The following steps are the usual "flow" for developing application programs with the LT Editor software. This lesson will follow those steps. (It is assumed that LT Editor is already installed.)

#### 1. Start the LT Editor

Start up the LT Editor software.

Select the type of LT and external devices you will use.

#### 2. Assign Variables to External I/O and Enable I/O

Use the Logic Program Editor's I/O Configuration feature to assign variable names (device addresses) to I/O terminal numbers.

#### 3. Create Internal Variables

Create the variables used for internal relays, registers, timers, and counters.

#### 4. Create the Logic Program

Use the Logic Program Editor software to create a logic program

#### 5. Create LT Screens

Use the LT Editor software's Screen Editor to create LT screens.

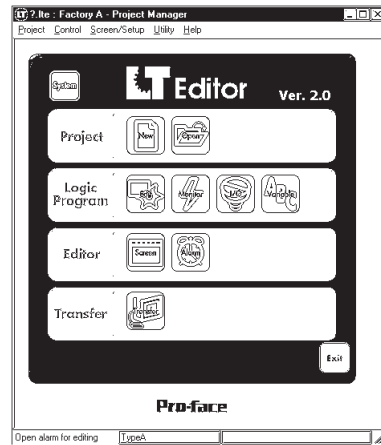
#### 6. Transfer Screens and Logic Programs to LT Unit/Check Operation

Transfer the screens and logic programs to the LT unit. Check that the LT operates correctly.

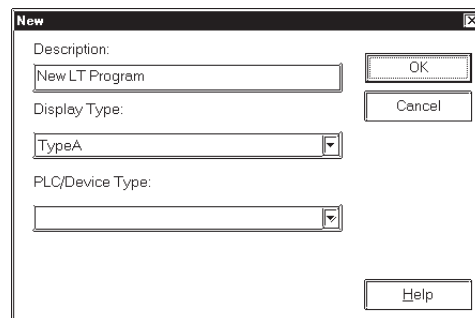
#### 7. Start "RUN" Mode Operation

## 1. Start the LT Editor

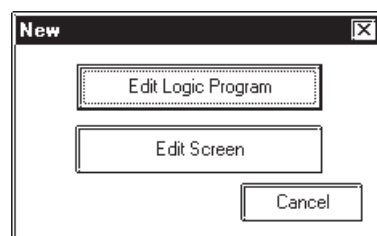
1. Click the Windows® desktop's [Start] button, and point to [Programs] → [LT] → [Project Manager].



2. Click the [New] icon. When the [New] dialog box appears, enter the following settings.
  - [Description] : New LT Program
  - [Display Type] : LT Type A
  - [PLC/Device Type] : None (Select only when using a Type-C unit.)



3. After all settings have been entered in the [New] dialog box, a second dialog box will appear. In this tutorial, click the [Edit LogicProgram] button to start up the Logic Program Editor and begin to create a logic program.



## 2. Assign Variables to External I/O and Enable I/O

With conventional PLCs, each PLC vendor uses their own naming system to handle External I/O addresses as I/O Device addresses. LT Editor, however, allocates arbitrary names to I/O Device addresses. These are referred to as variables.

These variables can be used for internal relays and timers, depending on the parameters that apply to the variable types and other settings used. The number of variables that can be created will depend on the memory variable area's size, and there are no special usage restrictions for individual variables.

### **Reference** 7.2 Variable Types

Use the Logic Program Editor's I/O configuration feature to assign variable names to external I/O.

#### 1. Setting up External I/O.

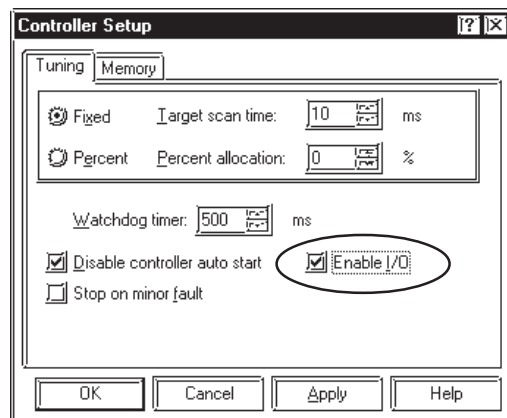
In the Logic Program Editor's [Controller] menu, select [Setup] and the [Setup] dialog box will appear.

Click the [Tuning] tab, select the [Enable I/O] check box, then click [OK].

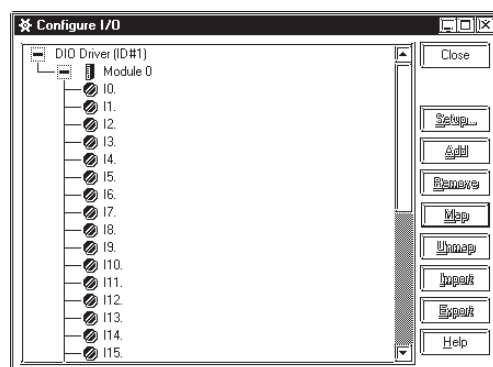


#### **Note:**

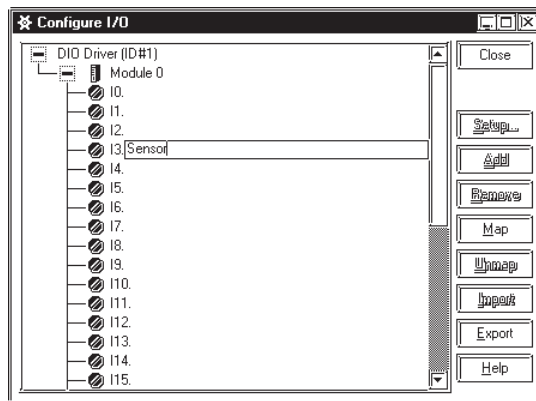
If the [Enable I/O] check box is not selected, external input/output will not be enabled, and only the LT's internal logic program will operate. (Can be used only for debugging.)



#### 2. To assign variable names [Sensor] and [Fan] to external I/O, click the [Data] menu's [Configure I/O] button and the [Configure I/O] window will appear.

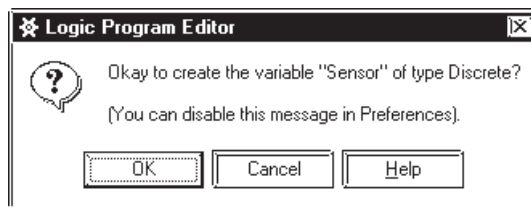


- I0 to I15 and Q0 to Q15 will appear below Module 0, which is under the DIO driver. "I" represents input, and "Q" represents the output signal's external I/O. Double-click on "I3", type "Sensor" in the text entry box and press [ENTER].



- When the following dialog box appears, click the [OK] button.

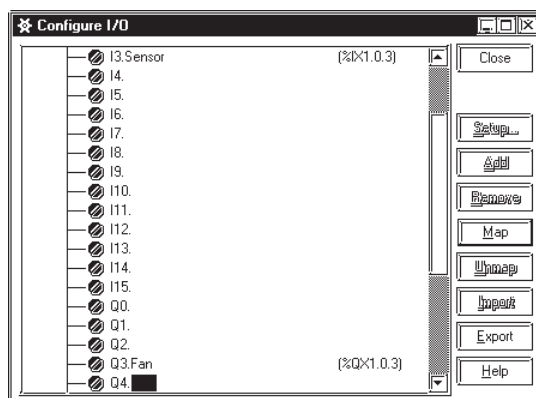
This will create a variable name for I3 and allocates that variable to the input terminal.



**Note:** “Discrete” indicates a variable type that uses bit units for processing.  
**Reference** [7.2 Variable Types](#)

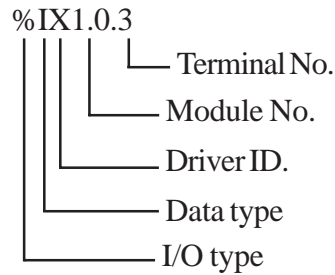
- Next, use the same procedure to assign the name "Fan" to "Q3". Variable names allocated here are used by the logic program and/or screen creation software to access external devices.

Here, we will assign “sensor” to a Normally-Open contact or a Normally-Closed contact instruction to receive input from the external input terminal. Similarly, output to the external output terminal can be performed by assigning “fan” to an OUT instruction.



**Note:**

When “%IX1.0.3” or similar characters appear in the I/O Configuration window, they indicate an I/O address. Each letter or symbol represents the following information:

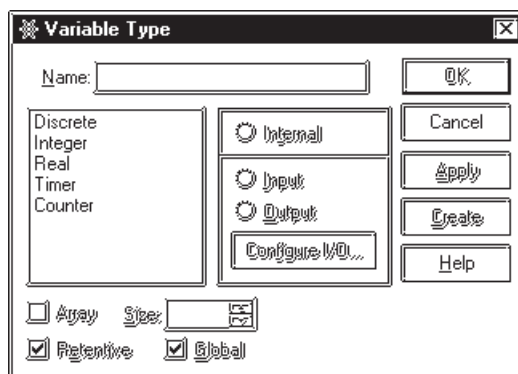


### 3. Create Internal Variables

Here, we will create named variables to be used for internal relays, registers, timers, and counters.

1. Let's create the variable “Run”, which will represent an internal relay.

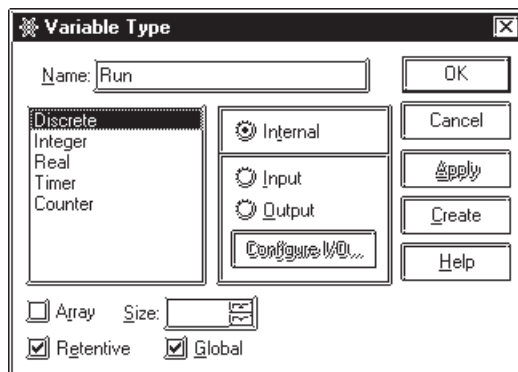
First, in the Logic Program Editor's [Data] menu, click [Variable Type] to call up the [Variable Type] dialog box.



2. Enter “Run” in the “Name” field, and select “Discrete” from the left-side Variable type menu, which processes data in bit units.


Select "Internal" to specify an internal variable, and click [OK].

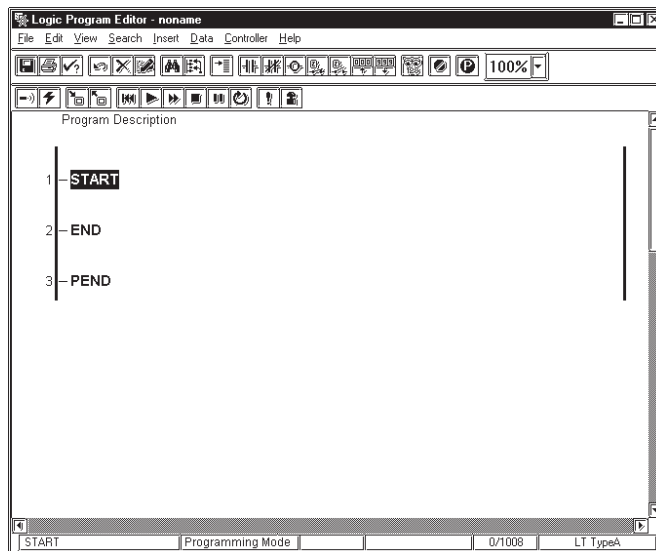
This creates the “Run” variable.






## 4. Create the Logic Program

A logic program can be created by simply inserting instructions in a rung.

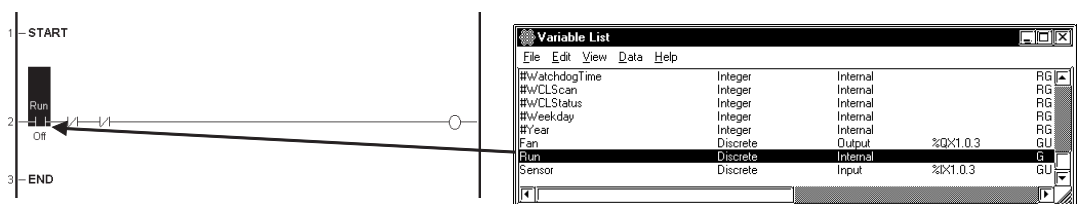
1. Click on [START] (Rung 1), and then click on the Tool Bar's  icon. Be sure to select [START] when creating the first rung.



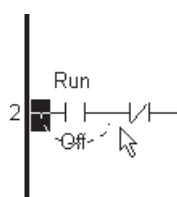
2. Click the Tool Bar's  icon to insert a Normally-Open contact in Rung 2. Next, on the same rung, click  twice to create two (2) Normally-Closed contacts. Last, click  to create a Coil.




3. From the Menu Bar's [Data] menu, click [Variable List]. Select "Run" from the list of variables that appear and, without releasing your mouse button, drag and drop "Run" to the far left-side Normally-Open contact.

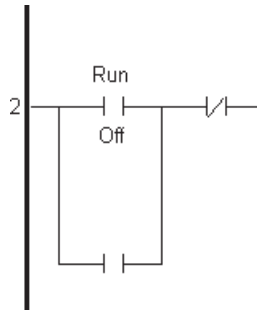


4. To create an automatic hold circuit, drag the "Run" circuit's left-side connection line to the right to create an OR circuit.

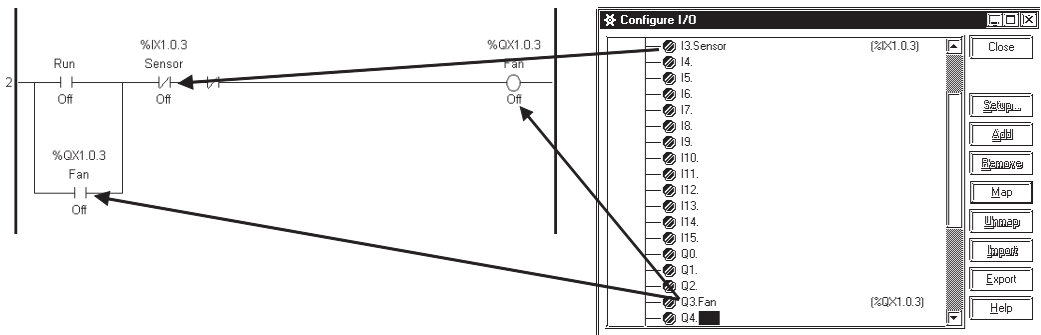



## Setup Guide (Tutorial)

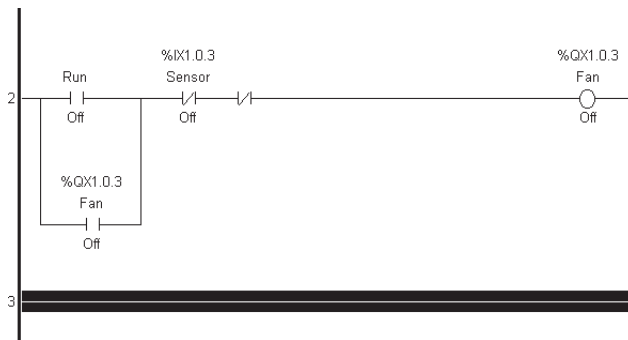
- Click on the lower branch of the OR circuit to select it, and click the  icon to insert a Normally-open contact.



- Next, we'll assign previously configured I/O variables to the logic program.  
First, click on "Sensor" and drag it to the left-most Normally-Closed contact. Repeat the process with "Fan", and drag it to both the OP circuit's Normally-Open contact, and to the Coil.



- To insert a new rung below Rung 2, click on the left side to select Rung 2 and then click on the  icon.



- Next, click on the  icon to create a Normally-Open contact on Rung 3 (see diagram).

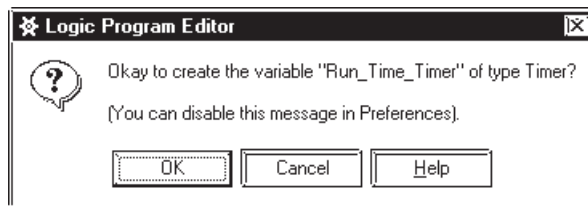




- Click on the  icon and create an On-Delay timer (TON) instruction. Enter the name "Run\_Time\_Timer" and press [Enter].

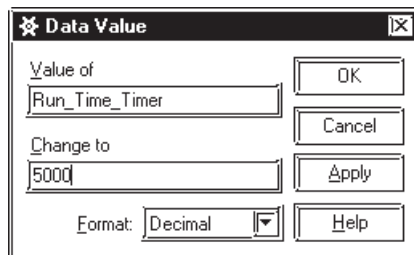


- When the variable confirmation dialog box appears, click [OK] to create the variable name "Run\_Time\_Timer". The name (variable) assigned to the On-Delay timer will be set to the "Timer" variable type. Use the same process to assign variables to contacts and coils.



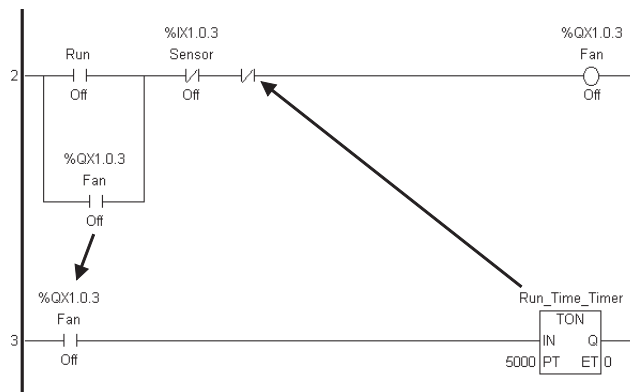
- Double-click the "0" field that appears in the lower left corner of the On-Delay timer to call up the [Data Value] dialog box.

Enter "5000" (milliseconds) in the "Change to" field to set the operating time to five seconds, and click [OK].



- Drag the variable name "Fan" from Rung 2 and drop it on Rung 3's Normally-open contact.

Next, drag Rung 3's "Run\_Time\_Timer" to the second Normally-closed contact in Rung 2 (as shown).



## Setup Guide (Tutorial)

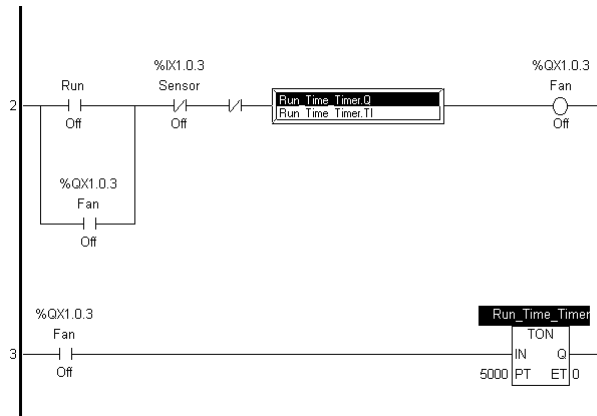
- After you drop "Run\_Time\_Timer" on Rung 2, a pop-up variable window will appear. Double-click on the window's [Run\_Time\_Timer.Q] variable, which designates the output bit used for "Run\_Time\_Timer".



**Note:**

The variable "Run\_Time\_Timer.Q" is a dedicated variable, and is created automatically when the Run Time Timer is created in step 10. This bit information (contact information) indicates that the time set in the Run Time Timer is elapsed.

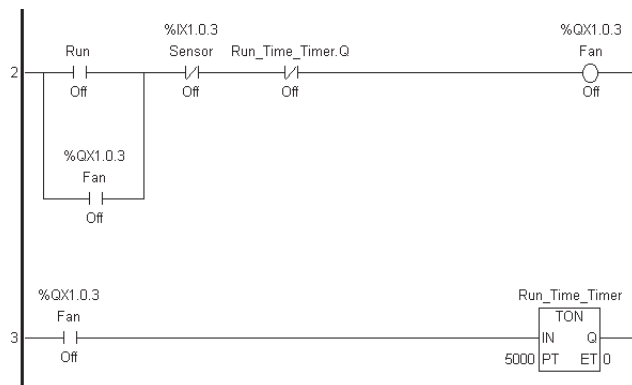
**Reference** 7.2 Variable Types ■ Timer and Counter



- Your logic program is now completed. Click on the Tool Bar's  icon to save this program.




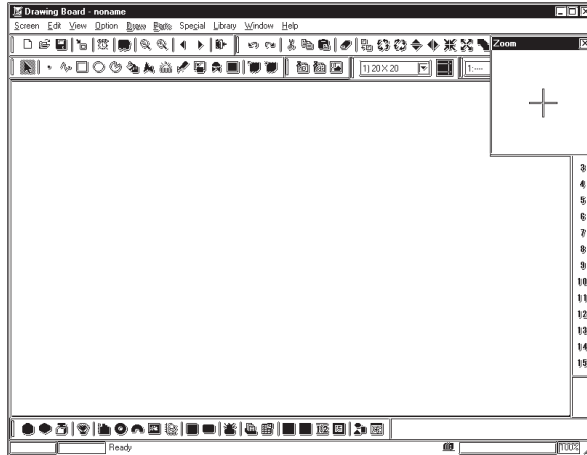
**This program's variable information is imported to the Drawing Board when the program is saved. Be sure to save your logic program before trying to create an LT screen.**



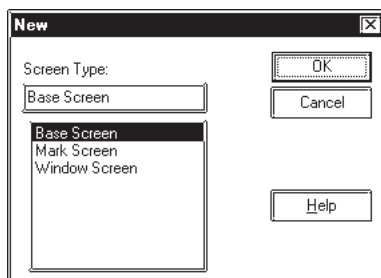
## 5. Create LT Screens

In this step we will create an LT display screen. Do not quit the Logic Program Editor when creating LT screens.

1. After saving your logic program, click on the Project Manager's [Screen] icon to call up the Drawing Board. Next, click on the Tool Bar's  (New) icon to create a new screen.



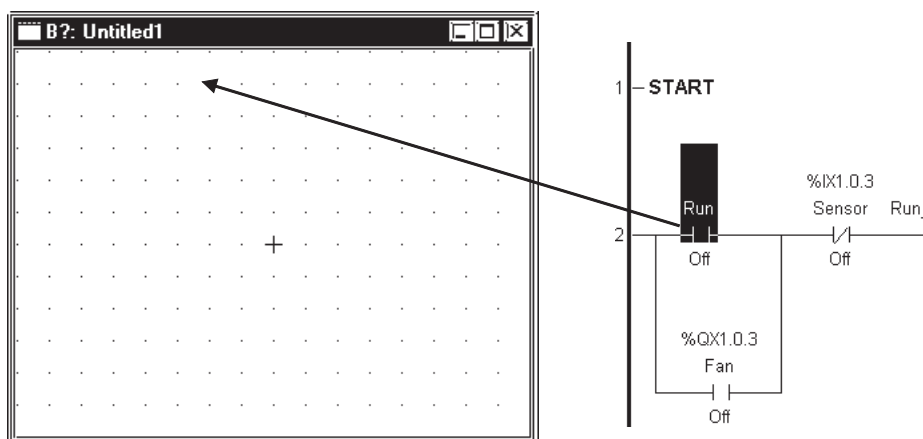
2. Select "Base Screen" as the screen type and click the [OK] button.



3. Click on the Logic Program Editor screen to activate it and drag "Run" (Rung 2 Normally-open contact) to the Drawing Board's base screen. Be sure to drag/select the entire command, not just the variable.



**You must save your logic program before dragging and dropping an instruction.**

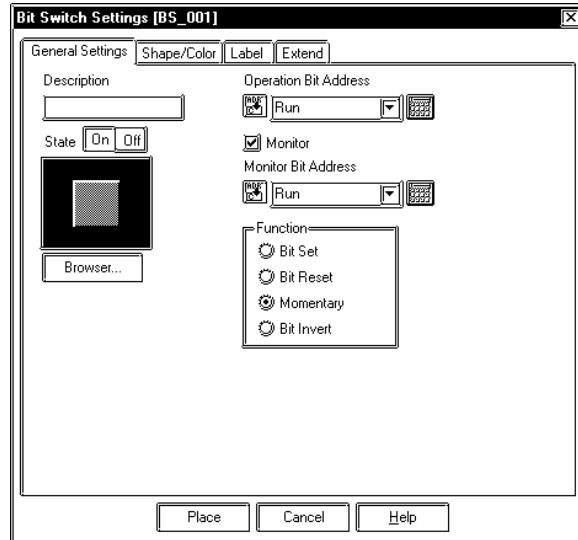


## Setup Guide (Tutorial)

- In the [Bit Switch Settings] dialog box's [General Settings] tab, select "Momentary" in the "Function" area. This function turns a bit (switch) ON only while the touch panel switch is touched.

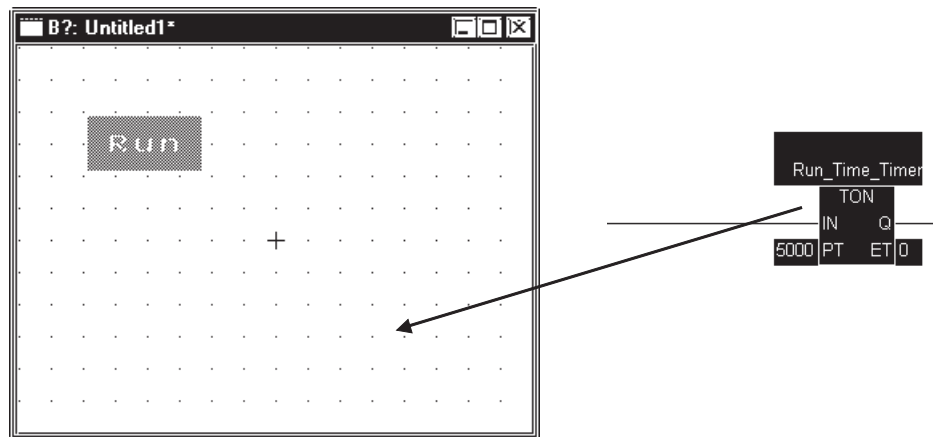
Next, click [Place] and position the bit switch on the base screen.

The switch's label (text displayed on the switch) and shape can also be set.

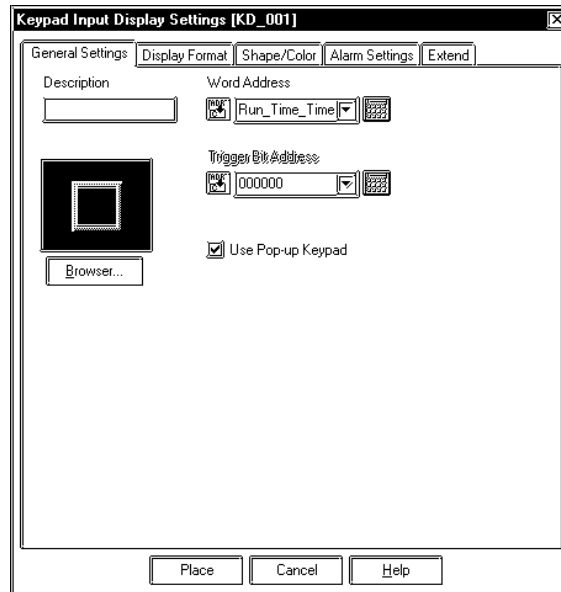


- Use the same procedure to drag the On-delay Timer (Rung 2) to the base screen.

The On-delay timer is treated as a Keypad Input Display when it is placed on a base screen.

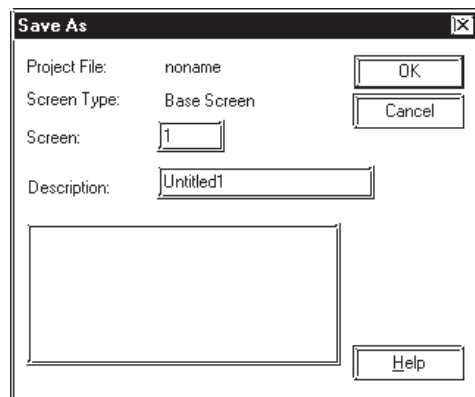


- When the [Keypad Input Display Settings] dialog box appears, click the [Place] button to place it on the screen. The Keypad Input Display, when touched, displays a keypad on the LT screen, allowing you to input numerical values.



- Next, click the Tool Bar's icon to call up the [Save As] dialog box. Enter "1" in the "Screen" field and click the [OK] button.

Setting the screen number to "1" designates that the screen will be used as the initial screen when the LT is started.



### 6. Transfer Screens and Logic Programs to LT Unit/Check Operation

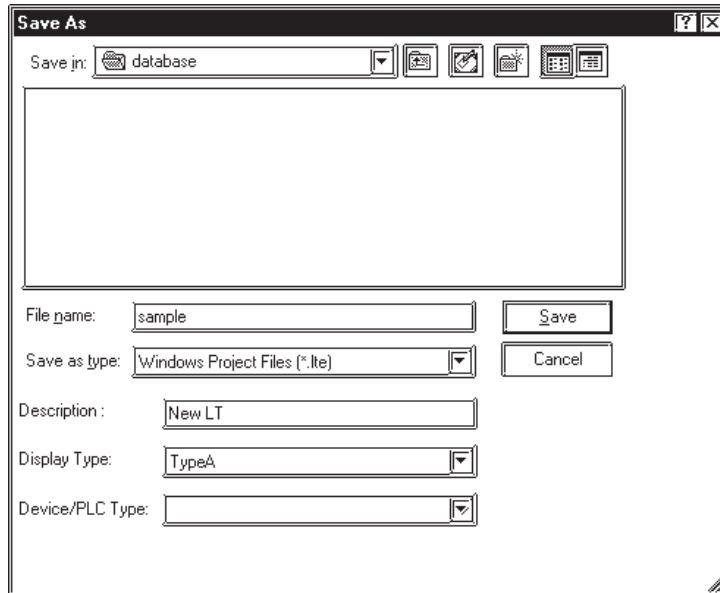
In this step, we will transfer the logic program and project screens we created to the LT to confirm that they operate correctly.

Prior to transferring the data, be sure to save your project (.lte) file.

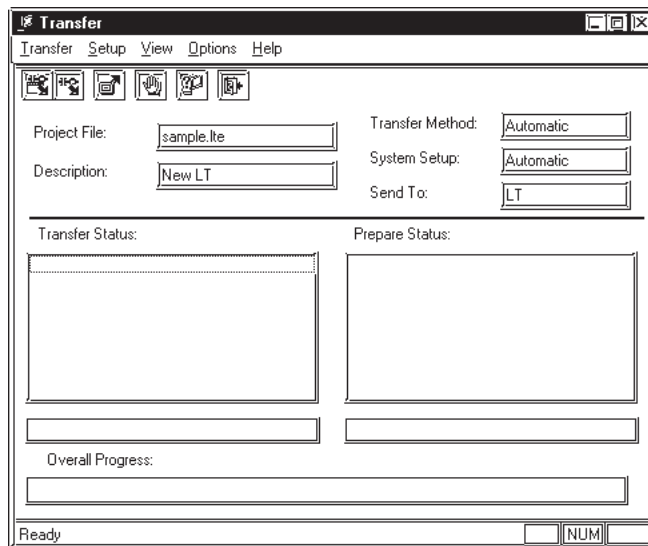
1. Quit the Logic Program Editor and close the Drawing Board.


Next, click on the Project Manager's [Project] menu and select [Save As].

Enter a file name and click [Save].

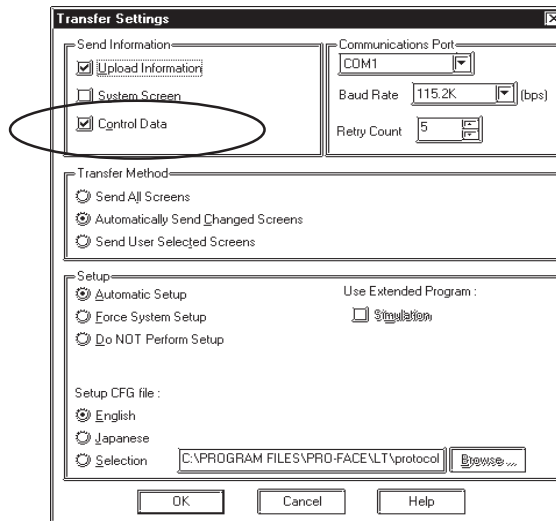



2. Click the Project Manager's [Transfer] icon and the following screen will appear.

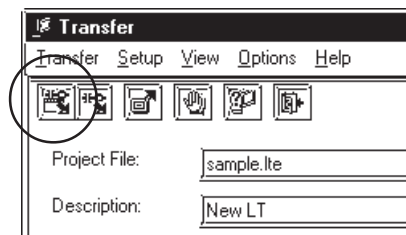


- Click the Tool Bar's  icon to call up the [Transfer Settings] dialog box. In the "Send Information" field, select "Control Data" and click [OK].

Next, select the desired PC port in the [Communications Port] field, and click [OK].



- Connect the data transfer cable to the LT, and click on the Tool Bar's  icon (to send both the screen and control data to the LT).



- After all data reaches the LT the unit is reset, and the screen you created is displayed.

Check that the screen and the logic program operate correctly.

The logic program created in this exercise should operate as follows.

- The fan rotates when "Run" is touched, and stops after five seconds.
- Touching the LT's Keypad Input Display displays a pop-up keypad on the screen, which allows you to change the stop time setting.
- If the sensor is activated while the fan is rotating, the fan stops.

This completes the sample program creation.

For detailed information on Logic Program Editor and Drawing Board operation, please refer to their corresponding manuals and Help data.

# *Memo*





# ***Programming***

# 1 Creating a Program

This chapter provides step-by-step instructions on using the Logic Program Editor to create a logic program in Programming mode.

For details on starting the Logic Program Editor, please refer to “1.2 Start to Finish” in the Operation Manual - Screen Creation Guide. For a detailed explanation of each part of the Logic Program Editor, please refer to the Features section and Online Help.

## ■ Before starting the tutorial

Each lesson in this chapter describes the operating procedures of the Logic Program Editor using tutorial examples. These practice examples are called “tutorials.”

This section describes how to use the Logic Program Editor to create a logic program that controls the operation of soft drink machines used in fast food restaurants. The machine features the following functions:

- Pressing the button once will automatically load a large/medium/small cup and dispense the required amount of soft drink.
- The ability to dispense ice or soda only if a cup is present under the dispenser.
- The ability to count the number of cups filled by the machine since it was powered on.

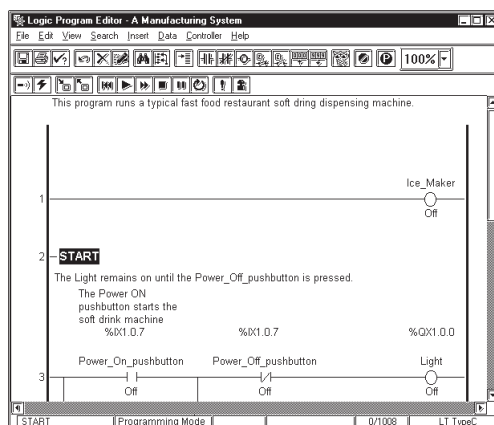
## ◆ Examples of Completed Logic Program and Screen

The logic program and project file used in this lesson can be found in the “Soda.lte” file, in the “C:\Program Files\Pro-face\LT\SAMPLE” folder.

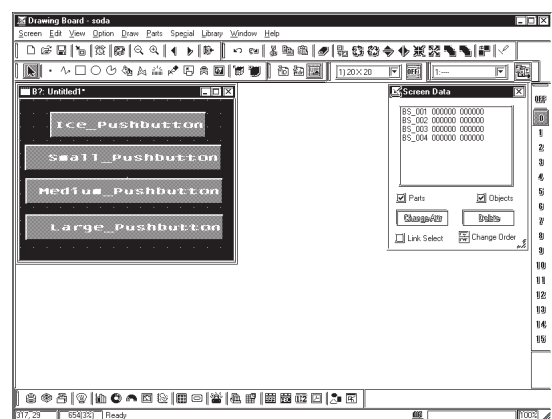
Refer to this file if you have problems with the procedure or wish to search for data items, or simply want to study.

**▼Reference▼** *Logic Program Editor Online Help.*

<Logic Program>

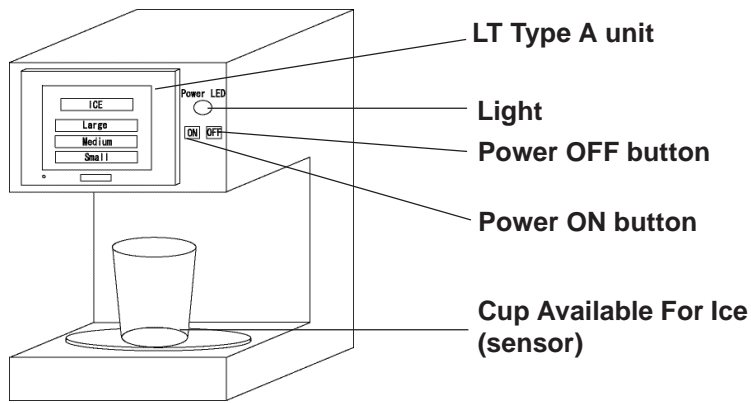


<Screen>



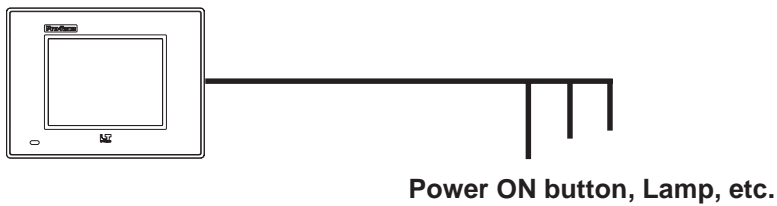
## Chapter 1 – Creating a Program

### ◆ Soft Drink Machine



### ◆ Hardware Design

#### LT Type A



### ◆ Allocating I/O Points

The “Ice\_pushbutton,” “Large\_pushbutton,” “Medium\_pushbutton,” and “Small\_pushbutton” are placed on the LT screen for touch-panel input and are therefore not allocated to a terminal.

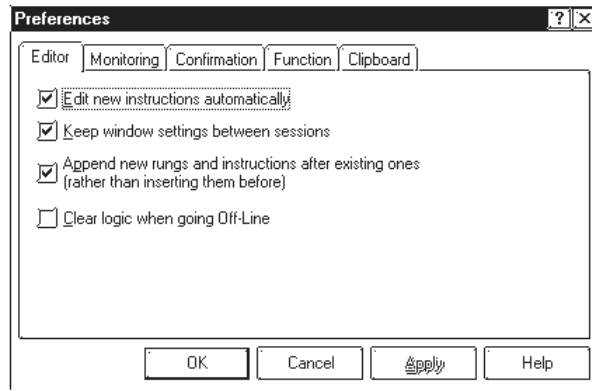
Variable Name	Terminal Type	Terminal No.
Power_ON_pushbutton	Input	I0
Cup_Present_for_Ice	Input	I2
Power_OFF_pushbutton	Input	I6
Light	Output	Q0
Ice	Output	Q1
Soda_valve	Output	Q2

## Preference Area Settings (Prior to Creating a Logic Program)

Prior to creating a logic program using the Logic Program Editor, you can designate the general settings used in order to customize your program creation/operation.

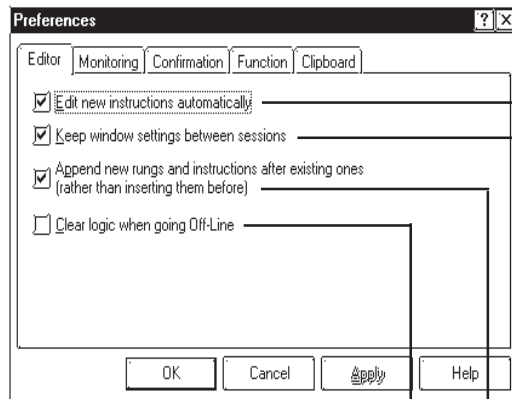
### ■ Designating Settings

1. Select [**P**references] from the [**F**ile] menu and the [**P**references] dialog box will appear.



2. Click on each check box to select or deselect a setting. The followings page's data explains each tab setting.

### ◆ Editor Tab



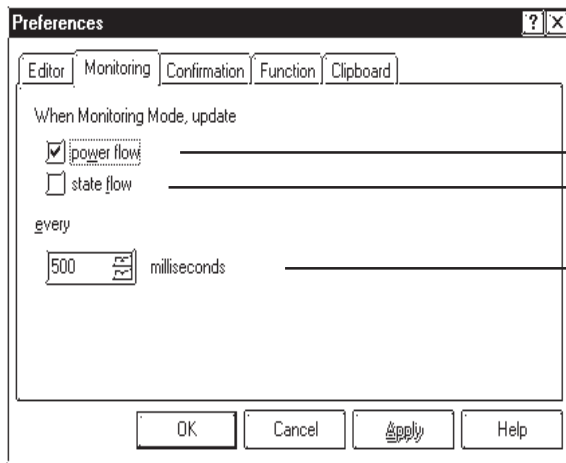
If selected, the [**I**nstruction **P**arameter] box is automatically opened for any new instructions inserted in your program. (Default: selected)

If selected, the Logic Program Editor opens all windows that were open at the end of the last session. Settings (such as window size and position) for any windows open during your editing session are retained. This also applies to the [**D**ata **W**atch] window which retains its contents when the current program runs online. (Default: selected)

If selected, new instructions are appended to the right of the [**f**ocus]. Objects (including rungs, labels, and subroutines) are appended below the [**f**ocus]. If cleared, new instructions are inserted to the left of the [**f**ocus]. Objects are inserted above the [**f**ocus]. If the [**f**ocus] is on a [**s**hunt], new instructions are inserted on the [**s**hunt]. (Default: selected)

If selected, the ladder logic screen will be cleared when going to Programming Mode from Monitoring Mode. (Default: not selected)

## ◆ Monitoring Tab



[**power flow**] is displayed while the Controller is in RUN mode.

The [**power flow**] highlights the display of the live (energized) rung (a vertical line used to describe instructions in logic programs) while the Controller is in RUN mode.

(Default: clear)

The [**state flow**] is displayed while the Controller is in the RUN mode.

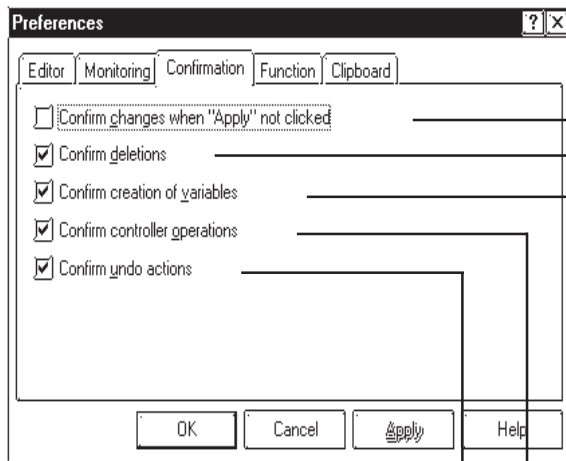
The [**state flow**] highlights the display of the live (energized) instruction while the Controller is in the RUN mode. The Power flow and State flow can be displayed at one time.

(Default: not selected)

Specifies how often the Logic Program Editor requests new data from the Controller to update [**power flow**], [**state flow**], data values, and the [**status bar**].

(Default: 500 ms.)

## ◆ Confirmation Tab



If selected, the Logic Program Editor accepts changes you make only when you click [**Apply**]. If cleared, the Logic Program Editor accepts changes immediately but asks for confirmation.

(Default: not selected)

If selected, the Logic Program Editor asks for confirmation for all deletions when you are creating your program.

(Default: selected)

If selected, the Logic Program Editor asks you to confirm the creation of every new variable in your program. This applies only to the Programming Mode environment.

(Default: selected)

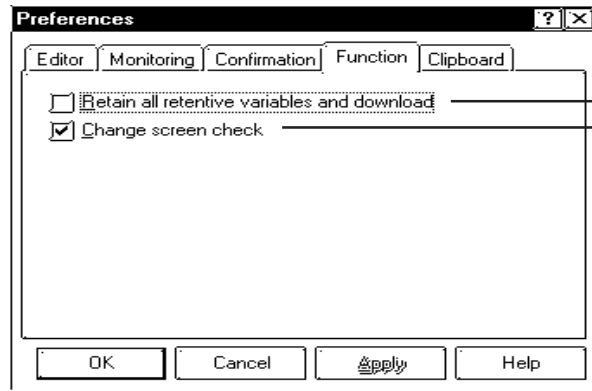
If selected, the Logic Program Editor asks you to confirm any change in the Controller operation (i.e., Start/Stop, Read/Write.)

(Default: selected)

If selected, the Logic Program Editor asks you to confirm any undo action.

(Default: selected)

◆ **FunctionTab**



If selected, retentive variable values will be retained when writing to the controller.

When the #Screen feature has been used to change a screen, after the change is completed, the #Screen value will be cleared to “0”.

In the logic program or the PLC, when a screen change is performed via the number entered in “#Screen” and “LS[8]” “LS0008”, “PLC’s Allocated Screen Change Number Device”, this feature allows you to check if the screen change has been completed or not.

**Retain all retentive variables and download**

(default: disabled)

When writing data to the controller, retentive variable values can be retained.

**Checked (enabled)**

All retentive variable values are retained. If the Confirmation tab’s [Confirm controller operations] is not selected, no confirmation message dialog box will be displayed.

**Not checked (disabled)**

All retentive variable values are initialized (set to “0”) when data is written to the controller.

**Change Screen Check**

(default: enabled)

This feature allows you to set whether the completion of a screen change is confirmed, when using the logic program or the PLC’s “#Screen” and “LS[8]” “LS0008”, [PLC’s Allocated Screen Change Number Device] to change screens via a set screen change number.

**Checked (Enabled)**

- When using Direct Access-

Zeroes (“0”) are written to “#Screen” and “LS[8]” “LS0008”, [PLC’s Allocated Screen Change Number Device] after the screen change has been confirmed (via comparing if the System Data Area’s currently displayed screen number is the same as the designated screen change number.).

- When using Memory Link

A “0” is written after the screen change has been confirmed (via comparing if the System Data Area’s currently displayed screen number is the same as the designated screen change number.).

**Not Checked (Disabled)**

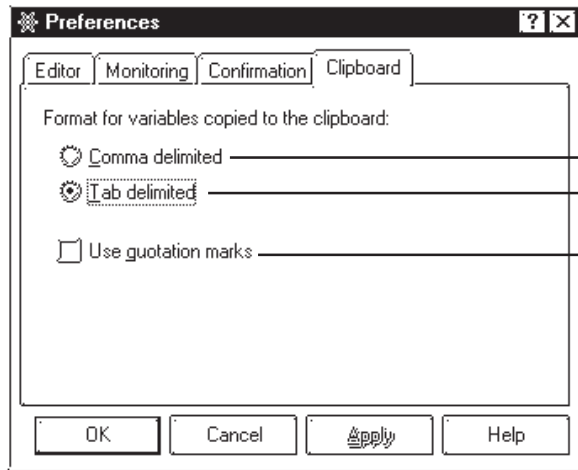
- When using Direct Access-

After the screen change has been confirmed, current screen change values are retained in the “#Screen” and “LS[8]” “LS0008”, “PLC’s Allocated Screen Change Number Device”.

- When using Memory Link

After the screen change has been confirmed, current screen change value is retained in “#Screen”.

### ◆ Clipboard Tab



- If selected, the fields copied from the variable list of the Logic Program Editor to the clipboard are separated by commas.  
Ex. My\_variable, Discrete, adescription  
(Default: not selected)
- If selected, the fields copied from the variable list of the Logic Program Editor to the clipboard are separated by tabs.  
Ex. My\_variable[TAB]Discrete  
[TAB] adescription  
(Default: selected)
- If selected, the fields copied from the variable list of the Logic Program Editor to the clipboard are separated by a delimiter and enclosed in double quotes.  
Ex. "My\_variable", "Discrete", "adescription"  
(Default: selected)

In this tutorial, be sure to use the default settings. Click on [Cancel] to close the [Preferences] dialog box and preserve the default settings.

### Exercise Overview

1. Start the LT Editor.

▼ **Reference** 1.1 *How to Start the LT Editor*

2. Select the LT and external device you use in the [New] dialog box.

▼ **Reference** 1.1 *How to Start the LT Editor*

3. Develop a logic program.

1. Determine variables.

This section describes how to designate the functions of the logic program to that is created by the Logic Program Editor as well as how to create and delete variables and set the initial values.

▼ **Reference** 1.2 *Creating and Variables*

2. Create a logic program.

This section describes how to create rungs, insert instructions and branches, and how to delete rungs, instructions and branches associated with the rungs.

▼ **Reference** 1.3 *Inserting Rungs, Instructions and Branches*

3. Assign variables to the logic program.

This section describes how to assign variables to the instructions in the logic program.

▼ **Reference** 1.4 *Assigning Variables to Instructions*

### 4. Insert descriptions.

This section describes how to label the logic program with descriptions. The description instructions include procedures for documenting the entire program, specific rungs, and individual instructions.

**▼Reference▲ 1.5 Documenting a Ladder Logic Program**

### 5. Edit.

This section describes how to copy, cut and paste rungs.

**▼Reference▲ 1.6 Copying, Cutting and Pasting Rungs**

### 6. Subroutine

This section describes how to insert subroutines and labels in the logic program.

**▼Reference▲ 1.7 Subroutines and Labels**

### 7. Search.

This section describes how to search and go to the desired circuit quickly in the logic program.

**▼Reference▲ 1.8 Navigating a Ladder Logic Program**

### 8. Assign I/O.

This section describes how to assign the logical variables in the logic program to the actual I/O terminals.

**▼Reference▲ 1.9 I/O Configuration**

### 9. Error check.

This section describes how to check for errors in the logic program.

**▼Reference▲ 1.10 Checking the Validity of a Program**

### 10. Print.

This section describes how to print out the logic program.

**▼Reference▲ 1.11 Printing Your Ladder Logic Program**

### 11. Import and export.

This section describes how to "read" and "write" the logic program.

**▼Reference▲ 1.12 Importing/Exporting a Logic Program**

### 4. Develop a screen program.

Use the Drawing Board and create a screen linked to the logic program.

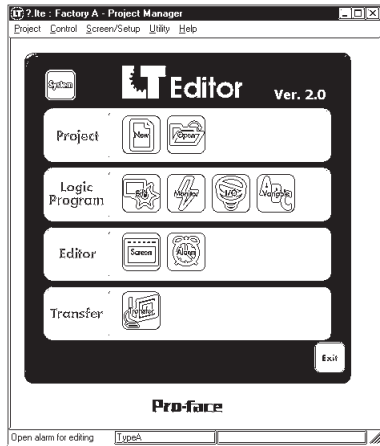
**▼Reference▲ 1.13 Developing a Screen Program**



# 1.1 How to Start the LT Editor

Activate the Project Manager prior to creating a logic program with the Logic Program Editor.

1. Click the **[Start]** button on the Window's screen, and point to **[Programs]** → **[LT]** and then click **[Project Manager]**.
2. The Project Manager starts up.

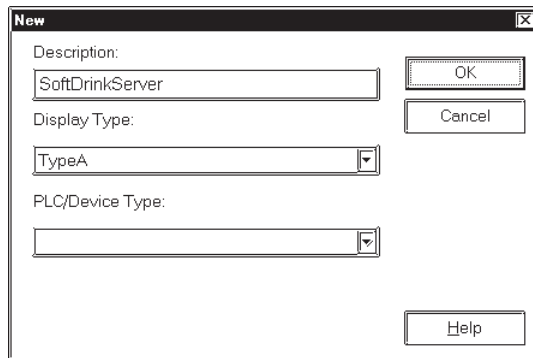


3. In the Project Manager screen, select **[New]** from the **[Project]** menu, or click the  icon. Input the settings as follows, and press the **[OK]** button.

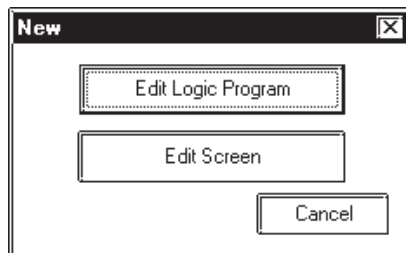
Description: Soft Drink Server

Display Type: TypeA

PLC/Device Type: None



4. A window appears asking whether you will create a Logic Program or Screen. Click **[Edit LogicProgram]** to activate the Logic Program Editor.



## 1.2 Creating Variables

This section describes how to designate the functions of the Logic Program Editor as well as how to create and delete variables and set the initial values used on the Logic Program Editor.

The completed sample of the tutorial program created in this lesson is located in the "Soda.lte" file in the "C:\Program Files\Pro-face\LT\SAMPLE" folder.

▼ **Reference** ▲ *Chapter 7 Variables*

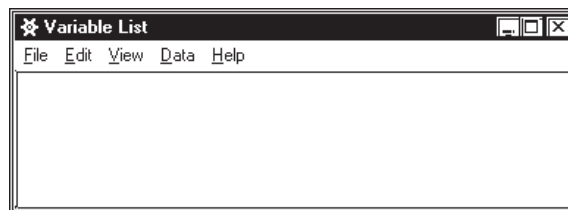
### 1.2.1 Creating a Variable List

You can add variables at any point while creating a ladder logic program. For convenience, create a list of the variables you will use in the tutorial now.

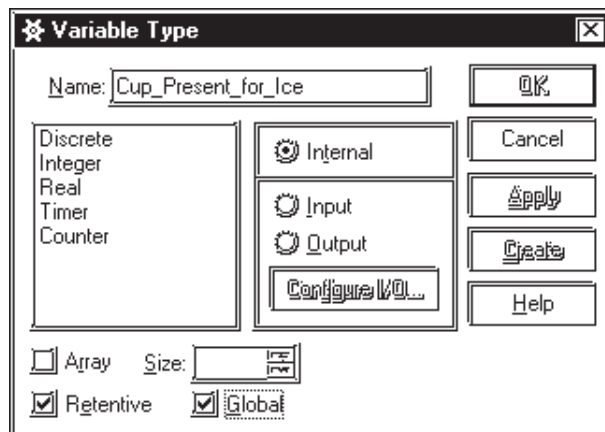
#### ■ Creating a List

Please refer to the online help for detailed descriptions of the menu items.

1. From the [**Data**] menu, select [**Variable List**]. The Variable List window is displayed.



2. From the [**Edit**] menu, select [**Add Variable**], and the [**Variable Type**] dialog box will appear.



3. Type "Cup\_Present\_for\_Ice" in the Name field.

▼ **Reference** ▲ *For details on variable name restrictions: Chapter 7 Variables*

**1.2.2 Selecting Variable Types**

The variable “Cup\_Present\_for\_Ice” is now displayed in the [Variable Type] dialog box. The words “Not Assigned” are highlighted in the list below it. There is no variable type assigned to “Cup\_Present\_for\_Ice”. Therefore, it needs to be assigned as a discrete input.

**Reference** For the variable types: 7.2 Variable Types

**■ Assigning Variable Types**

1. Select [Discrete] from the [Variable Type] list.
2. Select [Input].
3. Click on the [Retentive] box to deselect it. Data will not be retained if the power supply is cut, or the LT unit is reset.
4. Click on [Create]. “Cup\_Present\_for\_Ice” has now been assigned as a discrete input.

Note that the variable type change that you made to “Cup\_Present\_for\_Ice” in the [Variable Type] dialog box has now taken effect in the [Variable List] window and that the Variable Type dialog box is still open. If you had clicked on [OK], the changes would still have occurred in the [Variable List] window, but the [Variable Type] dialog box would have closed. The advantages of leaving these dialog boxes open becomes apparent as you begin inserting rungs and instructions as well as using LT Editor’s drag & drop, click, and insert features.



**Note:** You can select the variable types you want to view in the [Variable List] window by selecting [View], then selecting the variable types you want displayed. A check mark appears beside the selected variable types.

Now you have learned how to create a variable and assign a variable type to it, create the list of variables shown in the following table. Variables can be created directly in the [Variable Type] dialog box.

Variable Name	Variable Type	I/O Type	Hold/Release	Global
Power_On_pushbutton	Discrete	Input	Release	Local
Cup_Present_for_Ice	Discrete	Input	Release	Local
Ice_pushbutton	Discrete	Internal	Release	Global
Large_pushbutton	Discrete	Internal	Release	Global
Medium_pushbutton	Discrete	Internal	Release	Global
Small_pushbutton	Discrete	Internal	Release	Global
Power_Off_pushbutton	Discrete	Input	Release	Local
Ice	Discrete	Output	Release	Local
Soda_valve	Discrete	Output	Release	Local
Light	Discrete	Output	Release	Local
Fill_Timer	Timer	Internal	Hold	Local
Number_of_Larges	Counter	Internal	Release	Local
Number_of_Mediums	Counter	Internal	Release	Local
Number_of_Small	Counter	Internal	Release	Local

Close the [Variable Type] dialog box when you have finished.



**Note:**

If you typed a variable name incorrectly, simply rename it using the [Rename] option in the [Edit] menu's [Variable List] window. To create variables faster in the [Variable List] window, press the INSERT key.

### 1.2.3 Saving Your Program

To ensure the safety of created data, it is recommended that you save your logic program periodically. When a logic program is saved, global variables created with the Logic Program Editor are automatically registered to the Symbol Editor as Logic symbols, and can be used in common with the display function of the Drawing Board.

#### ■ To Save the Program

Select [Save] from the [File] menu on the Logic Program Editor screen.



**Note:**

You can also save your program by clicking  on the toolbar or by pressing the CTRL+S keys.

**Reference** *Operation Manual, Screen Creation Guide 4.2.5 – “Symbol Editor.”*

#### Summary

In this section you have learned how to:

- create variables and use dialog boxes associated with the variables
- determine variable types
- save a program

## 1.3 Inserting Rungs, Instructions, and Branches

The first step in creating a ladder logic program is to insert a rung. The screen initially shows a blank program as illustrated below. The completed sample of the tutorial program used in this lesson is located in the "Soda.lte" file in the "C:\Program Files\Pro-face\LT\SAMPLE" folder.



### 1.3.1 Inserting a Rung

Create a new logic program.

Down the left side of each new program are three rungs labelled START, END and PEND:

- The START rung indicates the start of the main program area.
- The END rung indicates the end of the main program area.
- The PEND rung indicates the end of the total program area. No rungs can be inserted after the PEND rung.

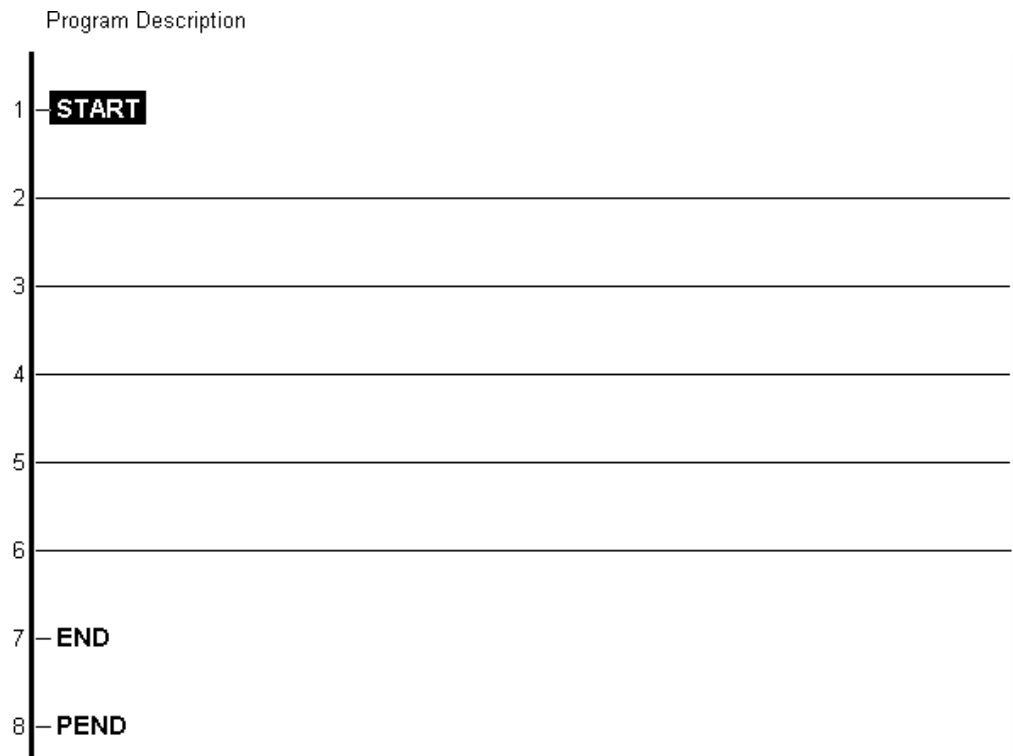
The rungs between START and END are executed every scan. Any rungs inserted in the area above START are for program initialization. This area is executed only during the first scan after power-up.


The area between the END and PEND rung is reserved for subroutines.

**Reference** Refer to the *Programmer's Reference* in the *Online Help* for a detailed explanation of the START, END, and PEND rungs.

#### ■ To Insert a Rung

1. Click on the rung number 1 left of the word START. Rung 1 is selected.
2. Right click once. A shortcut menu appears.
3. Select **[Insert Rung]**. (Or select **[Rung]** from the **[Insert]** menu.) A new rung appears at number 2, below the START rung.
4. Using the above method, insert four more rungs below the START rung. The screen will be like the picture shown below.

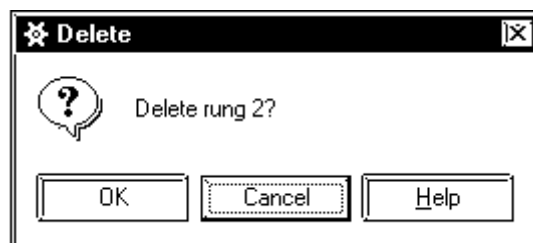


**Note:** You can also insert a rung by selecting [Rung] from the [Insert] menu, or by clicking on  in the toolbar.

### 1.3.2 Deleting a Rung


#### ■ To delete a rung

1. Select the rung you want to delete. In this example click on the number “2” (the rung number) on the left side of rung 2.
2. Press the DELETE key, or right-click on the rung and click on the **[Delete Rung]** selection. The **[Delete]** dialog box will appear.



3. Click on **[OK]**.



**Note:** As with other Windows applications, the LT Editor has an “Undo” command. From the [Edit] menu, select [Undo Changes to XX], or click on  in the toolbar.

**1.3.3 Inserting Instructions**

There are many ways to insert instructions into a ladder logic program and assign variables to them. As you create the ladder logic program in the tutorial, these methods are described and used.

**Reference** For detailed information about instructions, see *Chapter 7 Variables* or refer to the *Online Help*.

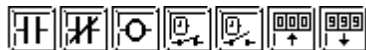
**Selecting a rung to insert instructions**

- Here, you are inserting instructions on rung 2. Click on anywhere on the rung 2 line to select it, but not on the number “2” itself. The selected rung will then be highlighted, as shown below.



- Once you have selected this rung, you can insert instructions. One way to do that is from the toolbar.

The Logic Program Editor toolbar contains the following buttons.



Click on these buttons to insert instructions into a selected rung. The meaning of these buttons is as follows.


	Normally Open Contact (NO)
	Normally Closed Contact (NC)
	Coil (OUT)
	Timer On Delay (TON)
	Timer Off Delay (TOF)
	Up Counter (CTU)
	Down Counter (CTD)

### ■ Method 1: Insert instructions from the toolbar

1. Click on the  button. The following box will appear.



The instruction now appears on the selected rung. Also, there is a box above it with a flashing cursor inside. This is the “Instruction Parameter Box” and is where you enter a variable to associate with the instruction. This will be explained in more detail later in this chapter.

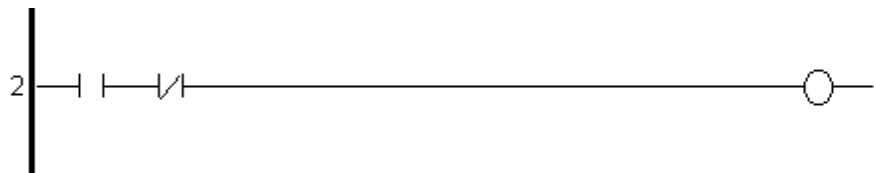
2. Click on the  button. This places an output coil on the right side of rung 2. Though the “Instruction Parameter Box” is still flashing, please ignore it for now.

**Reference** For Variable entry information, refer to *1.4 Assigning Variables to Instructions*.



3. Click on rung 2, between the **NO** and **OUT** instructions.

4. Click on the “Normally Closed” (NC) button , and that symbol will appear.

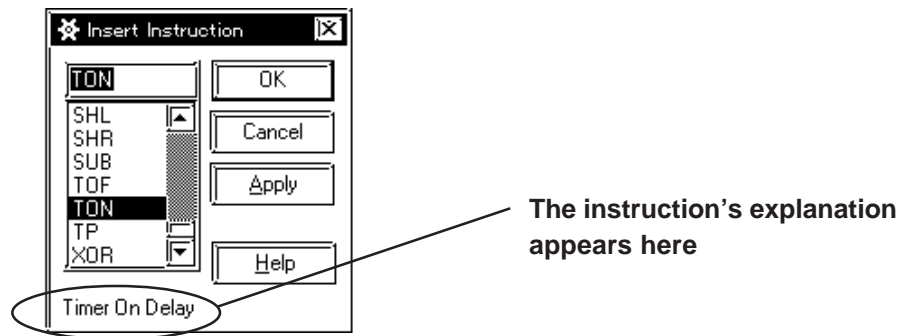


**Note:** For a description of each toolbar button’s feature, place the cursor over the button and read information that appears in the status bar. Though the toolbar offers an easy way to insert frequently used instructions, it does not include all Logic Program Editor instructions available within Logic Program Editor. You can also insert instructions from the [Insert Instruction] dialog box using the following two methods.



### ■ Method 2: Insert instructions from the [Insert Instruction] dialog box

1. Right click anywhere on rung 3 and a shortcut menu will appear.
2. Select [Insert Instruction]. The [Insert Instruction] dialog box appears.



This dialog box contains all instructions available to create a ladder logic program with the Logic Program Editor. As you type or click each instruction, a descriptor of the instruction appears at the bottom of the dialog box.



**Note:**

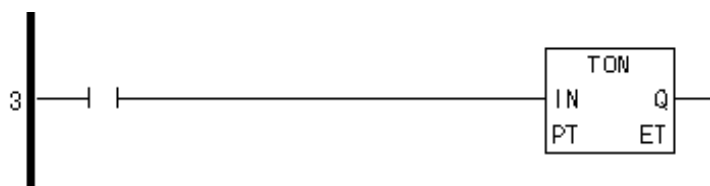
You can also bring up the [Insert Instruction] dialog box by selecting [Instruction] from the [Insert] menu or by pressing INSERT key after you have selected a rung. To view detailed information on each instruction, click the [Help] button while selecting the desired instruction.

3. Select the on-delay timer here. Scroll through the instruction list in the [Insert Instruction] dialog box until you locate the Timer On Delay (TON).
4. Select “TON”.

As with the [Variable Type] dialog box, you have a choice of clicking on either [OK] or [Apply] to register your selection. Since you are entering other instructions in your ladder logic program in this tutorial, the [Insert Instruction] dialog box needs to remain open. To do this, click on [Apply].



5. Click the left rung of the TON instruction.
6. Scroll through the instruction list in the [Insert Instruction] dialog box until you locate the Normally-Open contact (NO).
7. Double-click on “NO” and that symbol will appear.

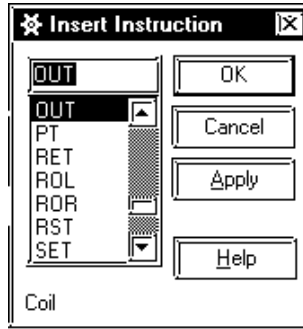


■ **Method 3: Insert instructions by typing in the [Insert Instruction] dialog box**

1. Type “out” in the field above the instruction list.



**Note:** The instruction list automatically scrolls until the “OUT” instruction appears at the top of the list. Also, its name appears in the bottom left hand corner of the dialog box.



2. Click on the rung section to the right of the TON instruction.
3. Click on [Apply] and the TON box will appear.

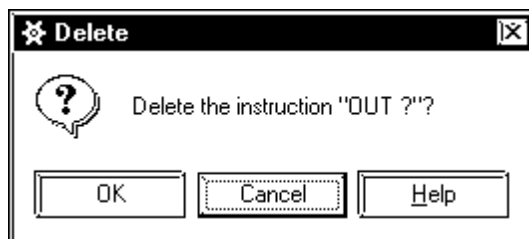


**1.3.4 Deleting Instructions**

Here, you will delete the OUT instruction you just inserted into rung 3.

■ **To delete an instruction**

1. Right click on the rung 3’s OUT instruction and a shortcut menu will appear.
2. Select [Delete]. A dialog box will appear to confirm that the instruction is to be deleted.



3. Click on [OK].



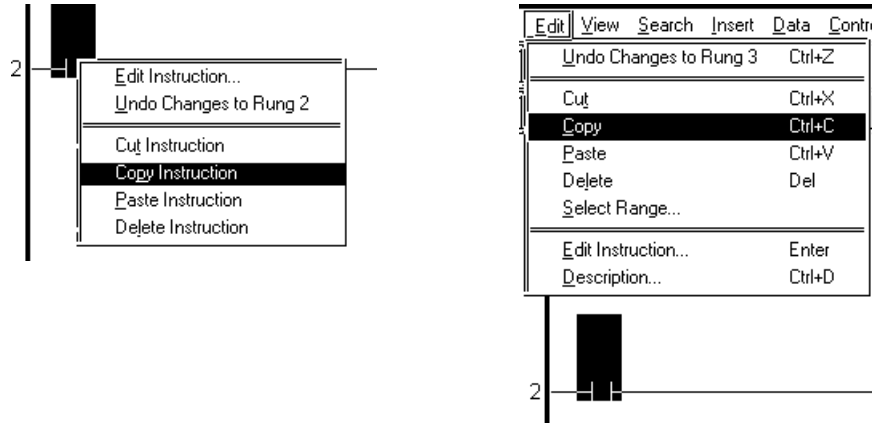
**Note:** You can also delete an instruction by selecting it and pressing the DELETE key, or clicking on  in the toolbar.

**1.3.5 Copying and Pasting Instructions**

Here, you will copy the instruction inserted into a rung and paste this instruction into another rung.

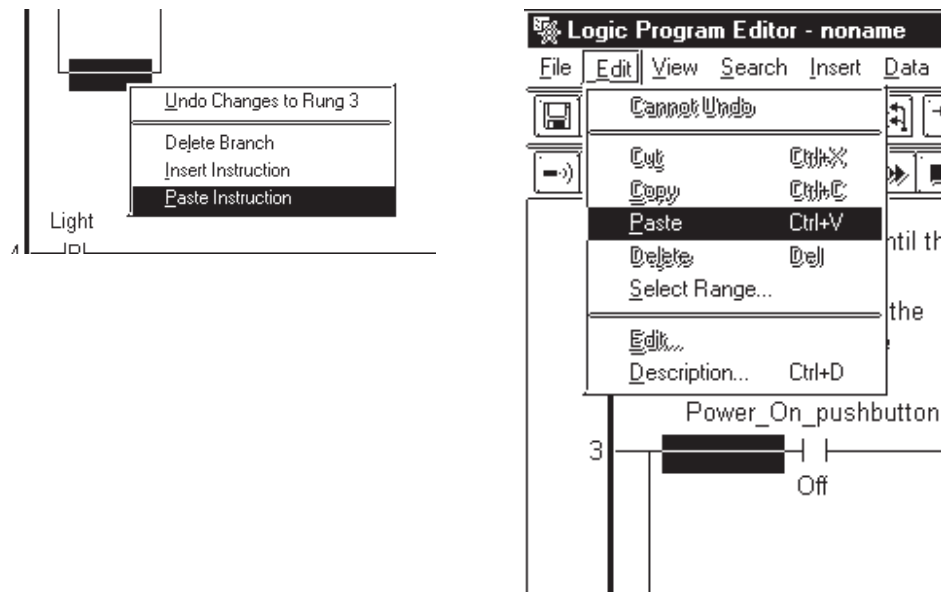
**■ To copy an instruction**

1. Click on the instruction you wish to copy.
2. Right-click and select [**Copy Instruction**], or select the [**Editor**] menu's [**Copy**].

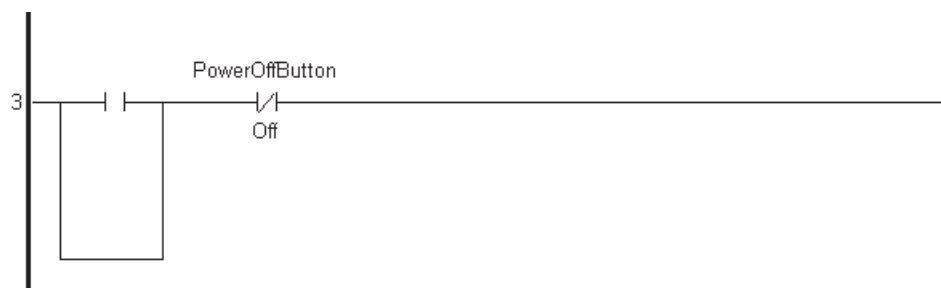


**■ To paste an instruction**

1. Click on the place you wish to insert the copied instruction.
2. Right-click on the [**Paste Instruction**] or click on the [**Edit**] menu's [**Paste**].




3. Now the copied instruction is pasted (inserted) into the desired rung.

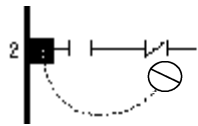




### 1.3.6 Inserting Branches


This section explains how you can insert a branch on rung 2 between the NO and the NC instructions. This branch is designed for the self-holding feature of the light on the soft drink server machine.

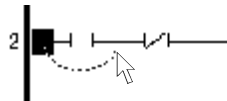
#### ■ To insert a branch

1. Place the cursor at the point on the rung where you want the branch to begin. In this case, directly to the left of the NO instruction.
2. Click and drag the mouse to the right. The cursor has turned into a  with a dotted line attached to it.

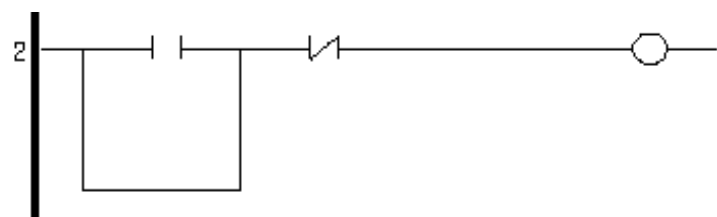


Whenever the end point of the branch is in an incorrect location, the Editor changes your cursor to a . Also, whenever the end point of the branch is in a valid location, the cursor returns to normal. If you release the cursor while it is normal, a branch is inserted between the starting point and where you released the mouse. If you release the mouse when the cursor is a , the branch will not be created.

3. Click and drag the mouse to the right until the cursor is between the NO and NC instructions and is not a .

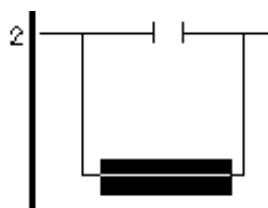


4. Release the mouse and a branch appears between the NO and NC instructions.



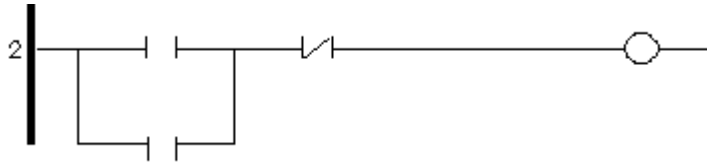
#### ■ To add an instruction to a branch

1. Select the branch by clicking on the bottom of it.



## Chapter 1 – Creating a Program

2. The **[Insert Instruction]** dialog box should still be open. If it is not, open it using any of the previously described methods
3. Select the NO instruction from the **[Insert Instruction]** dialog box and insert it using any of the previously described methods. Rung 2 will appear like this:



**Note:** To delete a branch containing instructions you must first select and delete each instruction.

### 1.3.7 Initialization Logic

Logic inserted above the START rung is called Initialization Logic. The logic program referred to in this section is executed only for the first scan when the Controller is started.

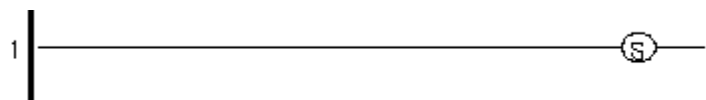
#### ■ To insert initialization logic

1. Right click on “**Program Description**” field located above the START rung. If it is not visible, select **[Descriptions]** from the **[View]** menu, and then select **[Program]**.
2. Select **[Insert Rung]** from the shortcut menu, and a rung is inserted above the START rung.



**Note:** In the following examples the rungs have been moved down one position (the rung which was previously number 2 is now rung 3).

3. Right click on the initialization rung (rung1).
4. Select **[Insert Instruction]** from the shortcut menu.
5. Select the SET instruction from the **[Insert Instruction]** window and click on **[OK]**.



This rung is used to turn the soda machine’s ice maker ON. It remains ON while the soda machine is started up and only needs to be set once.



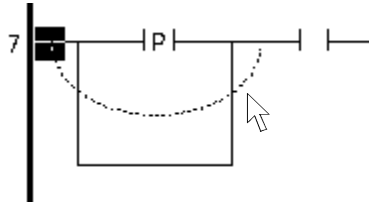
**Note:** If you do not have **[Append New Rungs and Instructions]** selected in the **[Preferences]** dialog box, you must select the START rung to insert any initialization rungs. These rungs will appear below the program description.

You have now completed rungs 3 and 4 of the ladder logic program as well as one rung of initialization logic. Please complete rungs 5-7, as shown on the following page.

Remember that the |P| instruction is a Positive Transition (PT) instruction.

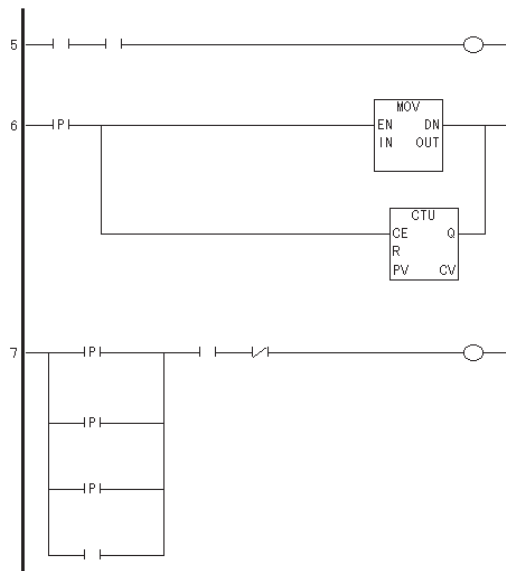
■ **To insert multiple branches into rung 7:**

1. Insert the first branch as previously described.
2. Insert the next branch by starting to click and drag from the same point as the previous branch.
3. Drag the cursor around the previous branch to the point on the rung where you want the branch to be inserted.



When the mouse is released, a new branch will be inserted over the previous branch, when is then pushed down.

In the example below, instructions have been inserted on rungs 5-7.



**Summary**

In this section, you have learned how to:

- insert and delete rungs
- insert and delete instructions
- insert and delete branches

## 1.4 Assigning Variables to Instructions

This exercise shows how to assign variables to instructions.

In 1.2 Creating Variables you created a variable list which includes some of the variables used in the tutorial ladder logic program. Please reopen the [Variable List] dialog box now.

### ■ To open the Variable List dialog box

1. From the [Data] menu, select [Variable List].
2. Move this dialog box to the lower left corner of your screen. If the [Insert Instruction] dialog box is still open, close it by clicking on [Cancel].

### 1.4.1 Instruction Parameter Box

In the previous section, a field appeared with a flashing cursor inside it when you first inserted an instruction on a rung. This is the **Instruction Parameter Box** and is where you enter the variables you want associated with the instruction.

### ■ To access the Instruction Parameter Box of a basic level instruction:

1. Double-click on rung 3's OUT instruction. A text field will open above the instruction with a flashing cursor inside of it. This is the "Instruction Parameter Box".



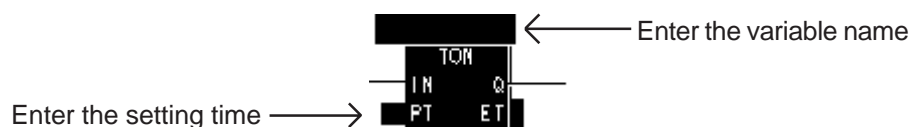
**Note:**

The "Instruction Parameter Box" can also be accessed by clicking on the instruction and pressing the ENTER key or by right clicking on the instruction and selecting [Edit Instruction] from the shortcut menu.

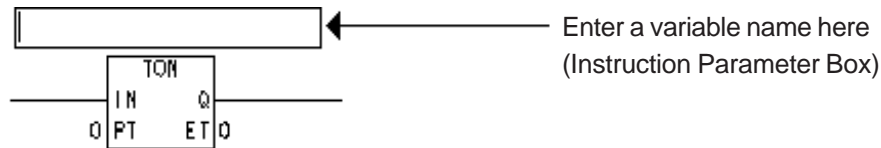
General instructions (non-basic level instruction) have more than one "Instruction Parameter Box". For example, a TIMER ON DELAY (TON) instruction has two (2). One is where you assign a variable, and the other is where you enter the preset time in milliseconds.

### ■ To access the Instruction Parameter Boxes of general instructions

1. Click on rung 4's TON instruction. The TON instruction then changes as follows:



Above the TON instruction a black highlighted area will appear. This is where you enter the variable to be assigned to the TON instruction. Next to the Preset (PT) element is another black highlighted area. This is where you enter the preset time in milliseconds.



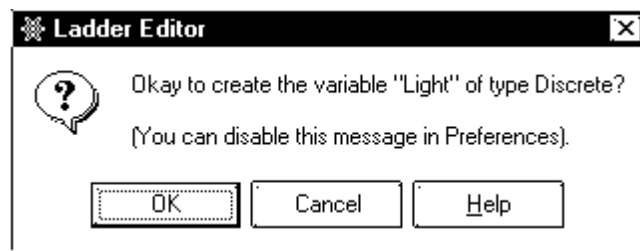
3. Next, double-click on the area immediately to the left of the PT element in the TON instruction. The **[Data Value]** dialog box opens. Here, enter the preset time in milliseconds that will elapse before output (Q) is turned ON. (Assigning variables and other operands to instructions will be discussed in the next section.)
4. Close the **[Data Value]** dialog box.

### 1.4.2 Entering Variables

One method of entering a variable into an Instruction Parameter Box is to type directly into the box.

#### ■ To enter text in the Instruction Parameter Box

1. Double-click on the OUT instruction's Instruction Parameter Box on rung 3.
2. Type "Light" in the box.
3. Press the ENTER key. The following dialog box appears asking you to confirm the creation of the variable.



4. Click on **[OK]**. In the **[Variable List]** dialog box the variable "Light" appears in the list. The Logic Program Editor has automatically assigned it a *variable type*. In this case it has assigned it as an internal discrete variable.



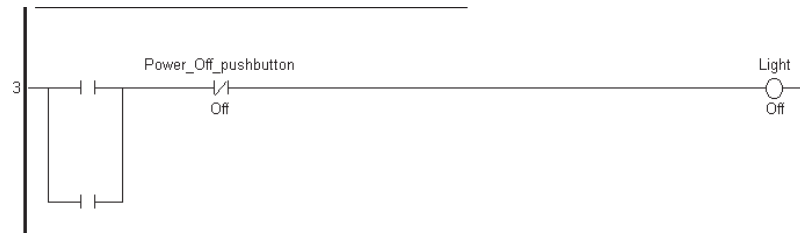
#### Note:

- The Logic Program Editor automatically assigns variable types to any new instruction variables created. You can also type a variable that already exists in your variable list directly into an Instruction Parameter Box. The variable is assigned automatically when you finished entering it.
- If you change the variables assigned to "Coil" instructions (i.e. OUT, SET, RST, NEG) to "Retentive", the "Coil" instructions also automatically change to "Retentive" type (i.e. M, SM, RM, NM).



## Chapter 1 – Creating a Program

Rung 3 should look like this:





Assign the variable “Ice\_Maker” to the SET coil on the first initialization rung. This variable can be created by typing it directly into the “Instruction Parameter Box”. After it is typed, the initialization rung appears as follows:

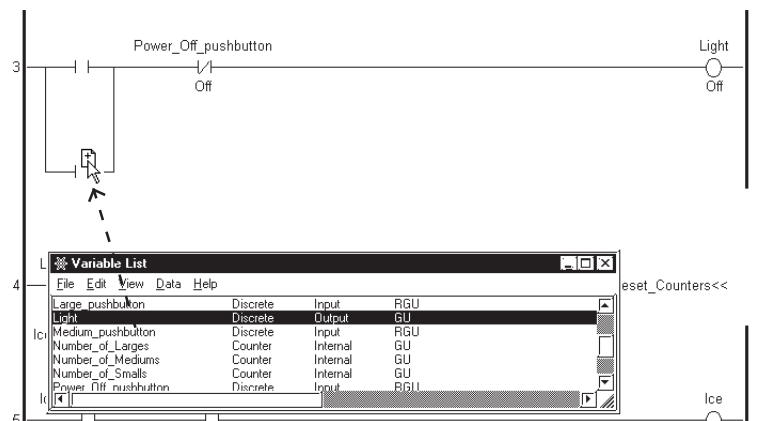



Another method of assigning variables to instructions is to simply drag the variable from the [Variable List] dialog box to the instruction itself. This method is very convenient if there are many instructions which need to have the same variables assigned to them. The advantages of using this method will be explained in Chapter 1.9 Assigning I/O.

### ■ To assign a variable using the Variable List dialog box

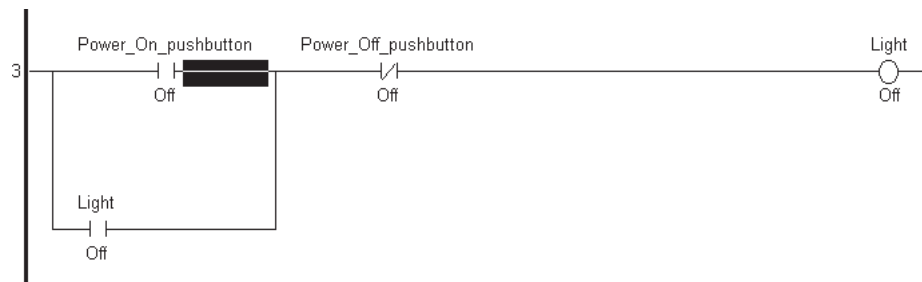
1. Call up the [Variable List] dialog box.
2. Click on “Light” in the [Variable List] dialog box but do not release the mouse button.
3. With the mouse button still pressed, drag “Light” to the NO instruction located on the branch on rung 3. As when inserting branches, note that your cursor initially becomes a . When the cursor is in this state you cannot assign the variable to any instruction.

When you research the No instruction, your cursor will change to a  mark.



The variable is then assigned when the cursor is released. As long as the cursor appears as a , you can assign the variable to an instruction.

4. Click on and then drag the “Power\_On\_pushbutton” variable to the other NO instruction on rung 3. Rung 3 should now appear as follows:



**Note:** In general, variables which are expressions, constants are assigned to instructions in exactly the same way as basic type variables, however, they must be typed in manually since there is no window to drag them from.

### 1.4.3 Completing the Program

Since you have learned how to assign variables to instructions, you can now complete the remaining rungs of the program. A diagram of the completed rungs is presented on the following page.

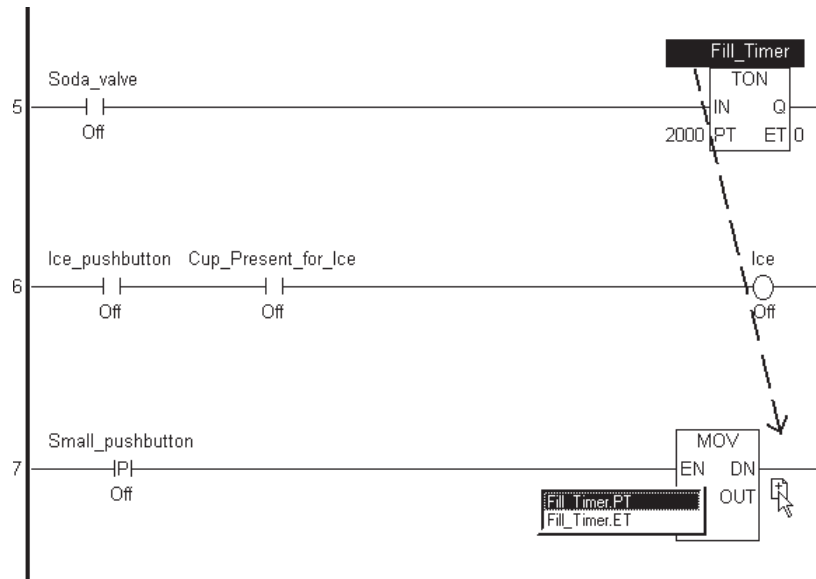
Notice that the MOV instruction on rung 6 and the NC instruction on rung 7 contain the variables “Fill\_Timer.PT” and “Fill\_Timer.Q” respectively. These variables refer to the “PT” and “Q” elements of the Timer with the “Fill\_timer” variable assigned to it.

The following three procedures are available for entering these variables. You can either:

- select the Instruction Parameter Box and type the “Fill\_Timer” variable in directly.
  - click on and drag the “Fill\_Timer” variable from the [Variable List] dialog box and add the “.PT” and “.Q” extensions in the Instruction Parameter Box.
  - drag the Instruction Parameter Box to the instruction you want to copy, and enter a variable selected from the special Variable List.
1. Select the source Instruction Parameter Box you want to copy from.
  2. Drag the counter and timer variables to the destination instruction you ant to copy.
  3. Select and double click on the desired parameter from the Variable List Box.

## Chapter 1 – Creating a Program

3. Select and double click on the desired parameter from the Variable List Box.



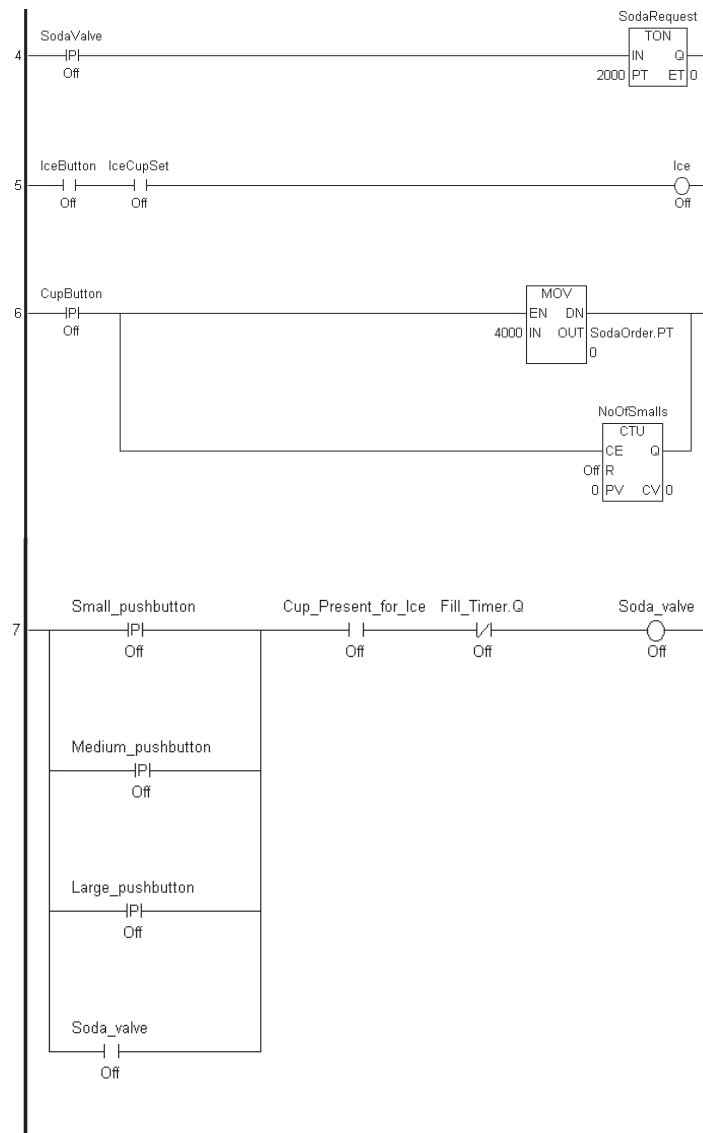
***These methods are used with rungs 6,7, and onwards. The application instructions' exclusive variables such as "Fill\_Timer.PT" or "Fill\_Timer.Q" consist of a variable name and a file extension:***

- \*\*\*.CV            (Current value)***
- \*\*\*.PT            (Set value)***
- \*\*\*.Q            (Output bit)***
- \*\*\*.R            (Reset bit)***

**▼ Reference ▲** 7.2 Variable Types

## Sample of Tutorial Program

The following logic program was created from the tutorial lessons so far.



## Summary

In this section, you have learned how to assign operands to instructions.

# 1.5 Documenting a Ladder Logic Program

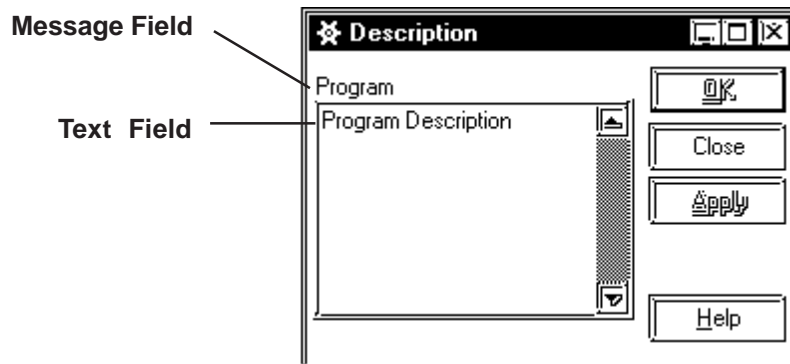
It is recommended that you document your ladder logic program. This data explains to users exactly how the program and each of its elements perform and is useful when the program needs to be altered or debugged later on. In the LT Editor, you can document how the program performs, how each rung operates and what specific variables are used for.

## 1.5.1 Adding a Program Description

The first description to add to your ladder logic program is an explanation of the program’s features.

### ■ To Add a Program Description

1. Double-click on the “**Program Description**” field at the top of the screen, and the [Description] dialog box will appear.

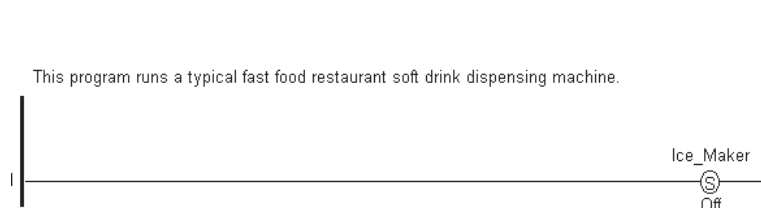


All LT Editor descriptions are entered here.



**Note:** The word “Program”, above the text field in the description dialog box, indicates that the text field contains a description of the program.

2. Click on the “**Program Description**” text.
3. Type “**This program runs a typical fast food restaurant soft drink dispensing machine**”.
4. Click on [OK]. This description now appears at the very top of the ladder logic program. (You may need to scroll up to see it.)



**Note:** You can also add or edit a Program Description by double-clicking the lower left-hand panel of the status bar.

## 1.5.2 Adding a Rung Description

Via the Logic Program Editor, you can add descriptions to each rung of your program. In the following example, a description is added to rung 5.

### ■ To add a rung description

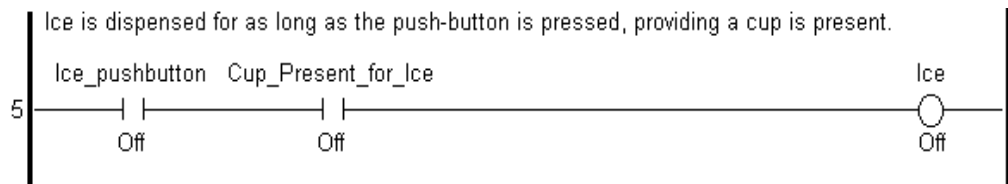
1. Right-click on rung 5's left side number.
2. Select [**Description**] from the shortcut menu and the [**Description**] dialog box opens. It is the same dialog box you opened previously, however, the descriptor above the text field now says **Rung 5** instead of **Program**.



**Note:** You can also open the [**Description**] dialog box by selecting [**Description**] from the [**Edit**] menu or by clicking on  in the toolbar.

Rung 5 controls the ice dispenser.

3. Click on the text field of the [**Description**] dialog box.
4. Type “**Ice is dispensed for as long as the push-button is pressed, providing a cup is present**”.
5. Click on [**Apply**].



To add descriptions to the remaining rungs of your program easily, keep the [**Description**] dialog box open.

### ■ To add a description to rung 3

1. Click anywhere on rung 3, outside of the **Instruction Parameter Boxes**. The descriptor at the top of the [**Description**] dialog box now says Rung 3.
2. Click on the text field.
3. Type “**The Light remains on until the Power\_Off\_Pushbutton is pressed**”.
4. Click on [**Apply**]. In this tutorial only the comments for rungs 3 and 5 are explained.

### 1.5.3 Adding Descriptions to Variables

Descriptions can also be added to each of the variables in your ladder logic program. You cannot however, add descriptions to labels or constants.

#### ■ To add a description to a variable

1. The [**Variable List**] dialog box should be open. If it is not, open it now by selecting [**Variable List**] from the [**Data**] menu.
2. The [**Description**] dialog box should also be open. If it is not, open it now by selecting [**Description**] from the [**Edit**] menu.
3. Click on any Instruction Parameter Box containing the variable “Fill\_Timer”. Note that not only does the [**Description**] dialog box contain the descriptor “Fill\_Timer”, but that “Fill\_Timer” is also highlighted in the [**Variable List**] dialog box.
4. Click on the text field of the [**Description**] dialog box.
5. Type “The Fill Timer decides how long to keep the soda valve open. The operating time depends on the set value”.
6. Click on [**Apply**].



**You can also add descriptions to a variable by selecting the variable in the [**Variable List**] dialog box, instead of selecting it from the ladder logic program.**

#### ■ To add a description

Here you will add a description to the variable “Power\_On\_pushbutton”.

1. Click on the variable “Power\_On\_pushbutton” in the [**Variable List**] dialog box. The [**Description**] dialog box now contains the descriptor “Power\_On\_pushbutton”.
2. Click on the text field of the [**Description**] dialog box.
3. Type “The Power On pushbutton starts the soft drink machine”.
4. Click on [**Apply**].

In this tutorial, descriptions are added to only the “Fill\_Timer” and “Power\_On\_Pushbutton” variables. Descriptions for other variables can be created by simply repeating the procedure described here.

## 1.5.4 Description List Dialog Box

The [**Description List**] dialog box displays brief, one line descriptions of all variables and rungs in the program.

### ■ To bring up the Description List dialog box

- From the [**View**] menu, select [**Description List**].

### ■ To view a detailed description from the Description List dialog box

Double-click the “Fill\_Timer” variable in the [**Description List**] dialog box. The [**Description**] dialog box displays the detailed description of the “Fill\_Timer” variable.

The [**Variable List**], [**Description**], and [**Description List**] dialog box displays change to reflect the rungs and variables selected in the ladder logic program. However, the opposite is not possible; for example, if a variable in the [**Variable List**] dialog box or a description from [**Description**] or [**Description List**] dialog boxes is selected, the corresponding choice is not reflected in the ladder logic. The search function of the Logic Program Editor allows you to find the specific variables easily. This will be explained in more detail in “1.8 Navigating a Ladder Logic Program.”

### Summary

You have learned how to add descriptions to the program, to rungs and to variables as well as how to call up the [**Description List**] dialog box.



## 1.6 Copying, Cutting and Pasting Rungs

When creating a ladder logic program, you may find you have to duplicate sequences of instructions on several rungs. You can speed up your work by copying and pasting completed rungs.

### 1.6.1 Copying a Rung

In the following exercise, two rungs are added between rungs 5 and 7. These additional rungs contain the same instructions as rung 6 with different variables assigned to them.

#### ■ To copy a rung

1. Click on the number “6”, shown on the left of the rung, to select entire Rung 6.
2. From the [Edit] menu, select [Copy].



**Note:** If you wish to select a range of rungs to be cut or copied, click on the rung number of the first rung you wish to select. Hold the [SHIFT] key down and select the rung number of the last rung you wish to select. All rungs between the two are then selected and can be cut or copied. Copying is limited to approximately 25 rungs.

### 1.6.2 Pasting a Rung

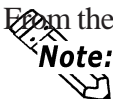
The Logic Program Editor pastes rung(s) below the current rung, as long as all the current rung is not selected. If [Append new rungs and instructions] is not selected in the [Preferences] dialog box, the copied rung is inserted above the current rung.



**Important** *A rung cut and pasted is loaded to the LT Editor’s internal clipboard, then copied to the program. If you select an entire rung when pasting from the clipboard, the Logic Program Editor replaces the rung you have selected with the rung in your clipboard.*

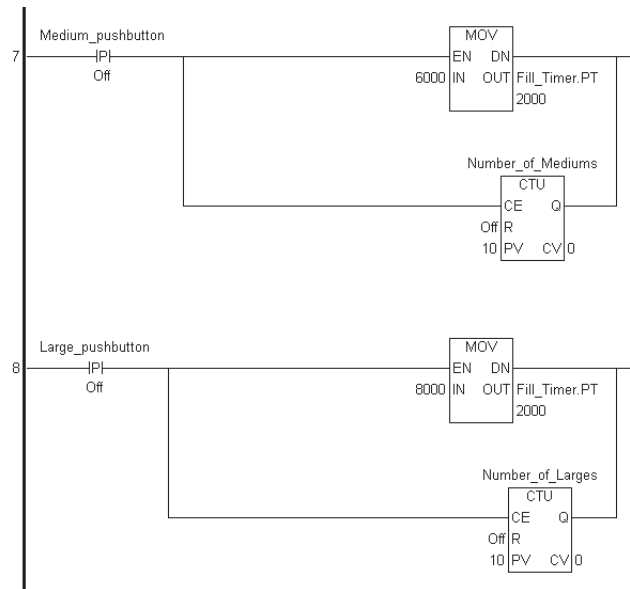
#### ■ To paste a rung

1. Click anywhere on rung 6.
2. From the [Edit] menu, select [Paste]. Rungs 6 and 7 are now identical.
3. Click anywhere on rung 6.
4. From the [Edit] menu, select [Paste]. Rungs 6 to 8 are now all alike.



**Note:** When pasting a rung, all variables and descriptions associated with that rung are also pasted. Be aware that you may have to edit the pasted rung.

The variables on rungs 7 and 8 should now be changed, according to the following example.



5. Change the variable name of the PT instruction on the rung as shown in the example above.

### 1.6.3 Cut Command

The Logic Program Editor’s Cut command allows you to take a rung or section of rungs out of one part of your program and move them to another. In the following tutorial, rung 4 is to be moved to the last rung of your program.

#### ■ To use the “Cut” command

1. Click on rung 4. The entire Rung 4 is selected.
2. From the [Edit] menu, select [Cut]. The rung is now taken from the ladder logic program and placed on the clipboard.
3. Click anywhere on rung 8.
4. From the [Edit] menu, select [Paste]. Rung 4 is now appended to below rung 8. The end of the program now appears as follows:



**Note:** To move an entire rung to another part of the program, first select the rung and drag it using the middle of the rung to the new location.

### Summary

In this section, you have learned how to copy, cut, and paste rungs.

## 1.7 Subroutines and Labels

When a **[JSR]** (jump to subroutine) or **[JMP]** (jump) instruction is inserted in a rung, it tells the Controller to resume scanning starting at that subroutine or label. The main difference between a subroutine and a label is that Editor executes a subroutine and then returns to the point in the ladder logic directly after the **[JSR]** instruction. If Editor jumps to a label (through the use of the **[JMP]** instruction), it continues executing the ladder logic program at that point and does not return to the **[JMP]** instruction during that scan.

**▼Reference** For more information on the **[JMP]** and **[JSR]** instructions, see the **9.2.41 JMP (jump)/ 9.2.42 JSR (jump to subroutine)**.

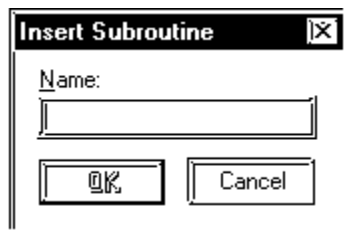
### 1.7.1 Inserting a Subroutine

At the bottom of every LT Editor program are two rungs labelled “**END**” and “**PEND**”. The “**END**” label signifies the end of the main program area. The Logic Program Editor executes the instructions between “**START**” and “**END**” with every scan. The area between the “**END**” label and the “**PEND**” (Program End) label is reserved for subroutines.

In the following tutorial, a subroutine is added.

#### ■ To insert a subroutine

1. Click on the **[END]** label.
2. From the **[Insert]** menu, select **[Subroutine]**. The **[Insert Subroutine]** dialog box appears.



A maximum of 32 characters, numbers, or underscore characters, can be used for a subroutine name. Variable names cannot begin with numerical characters and cannot contain spaces.

**▼Reference** **1.2.1 Creating a Variable List**

4. Click on **[OK]**. At the end of your program the subroutine will appear.

```

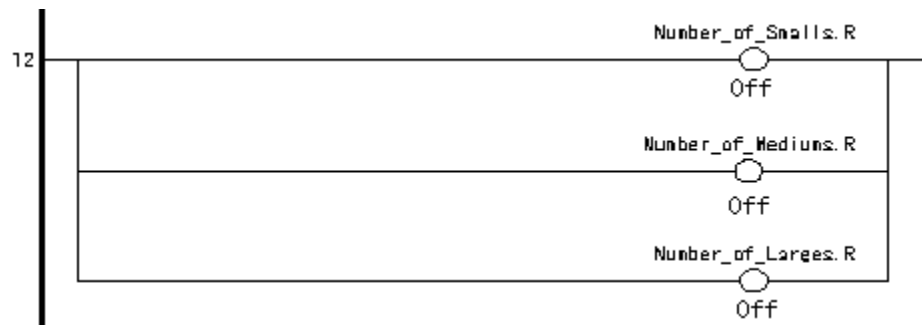
10— END
11— SUB STARTReset_Counters
12— SUB ENDReset_Counters
13— PEND

```

Here, you insert your subroutine between the two new rungs labelled “SUBSTARTReset\_Counters” and “SUBENDReset\_Counters”.

5. Right click on the “SUBSTARTReset\_Counters” label.
6. Select [**Insert Rung**] from the shortcut menu to insert a rung between the “SUBSTART” and “SUBEND” rungs.
7. Right click on the rung “SUBSTART” and “SUBEND”.
8. Insert an “OUT” instruction in the rung.
9. Insert 2 branches around the “OUT” instruction.
10. Insert an “OUT” instruction on each branch. The following is the completed subroutine.

This routine will reset each of the Counters every time the LT is turned ON.



Each of the variables you see here should be assigned to each of the “OUT” instructions. Assign these variables now.

This completes the subroutine you can add more than one subroutine to a ladder logic program by selecting either the “SUBSTART” or “PEND” rungs and repeating steps 2 through 6.

If you want a subroutine to be executed at some point in your ladder logic program you must insert a [**JSR**] instruction. This is explained in the following tutorial.

This subroutine is executed as soon as the ‘Light’ **OUTPUT COIL** on rung 3 turns ON. Therefore, the [**JSR**] instruction must be placed on rung 4.

### ■ To insert a [**JSR**] instruction:

1. Select rung 3.
2. From the [**Insert**] menu, select [**Rung**].
3. Insert a [**PT**] instruction on rung 4.
4. Assign the variable ‘Light’ to the [**PT**] instruction.
5. Insert a [**JSR**] instruction to the right of the [**PT**] instruction. This is done from the [**Insert Instruction**] dialog box.
6. Type ‘Reset\_Counters’, the name of the subroutine, in the [**Instruction Parameter Box**] of the [**JSR**] instruction. The rung appears as follows:



## Chapter 1 – Creating a Program

Whenever the [JSR] instruction “Reset\_Counters” receives power, it will jump to the subroutine “Reset\_Counters”. Execution will resume from rung 5 once the subroutine has finished execution.



### Note:

To delete a subroutine, you must first delete the individual rungs. After that, delete the “SUB START” rung. The “SUB END” rung will then be automatically deleted when the “SUB START” rung is deleted.

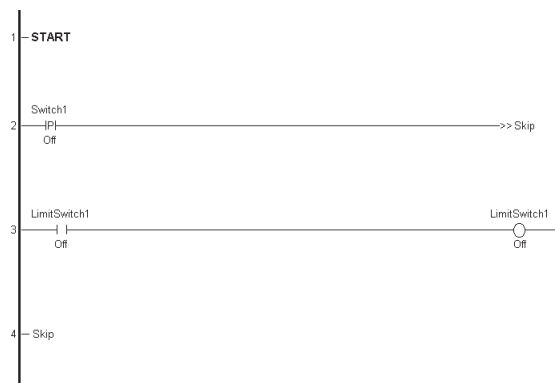
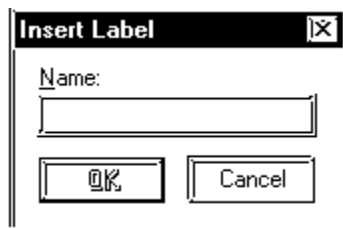
### 1.7.2 Inserting Labels

A label, which is combined with a [JMP] (Jump) instruction, can be inserted in any part of a ladder logic program. When the Controller executes a [JMP] instruction, it jumps to the designated label and begins executing the program at that point.

Labels are inserted above or below the selected rung depending if [Append new rungs and instructions] is selected in the [Preference] dialog box. This tutorial does not use any labels. However, to insert one, the following procedure is used.

#### ■ To assign a label to your ladder logic program:

1. Click anywhere on the rung.
2. From the [Insert] menu, select [Label]. The [Insert Label] dialog box appears prompting you to insert a name for your label.



This is the name that is designated in the [JMP] instruction in your ladder logic program. The same rules that apply to naming variables apply to naming labels.

#### ■ To insert a [JMP] instruction

1. Right click on the right of the last instruction on the rung and select [Insert Instruction] from the shortcut menu.
2. Double click the [JMP] instruction in the [Insert Instruction] dialog box. The [JMP] instruction is inserted as the last instruction on the rung. Whenever the Logic Program Editor sees this instruction in your program, it jumps to the designated label.

### Summary

This section explained how to create subroutines and labels and insert [JMP] (jump) and [JSR] (jump to subroutine) instructions.

## 1.8 Navigating a Ladder Logic Program

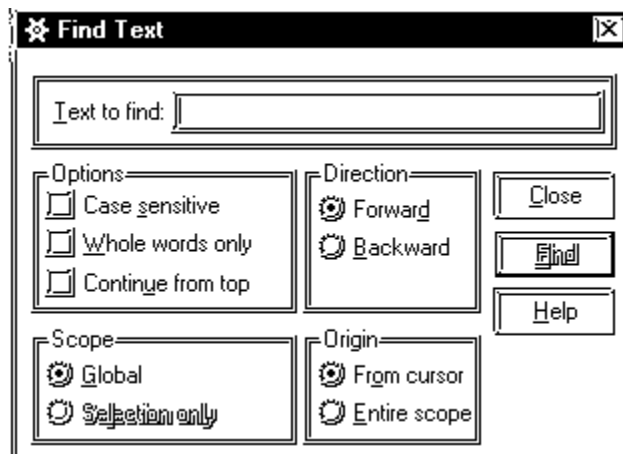
If a logic program is large, using the scroll bars to locate specific points in the program can take quite a bit of time. The Logic Program Editor features commands to help you find specific points in your program much more quickly. These are the **[Find]**, **[References]**, **[Bookmark]**, **[Go to Rung]** and **[Go to Label]** commands.

### 1.8.1 The **[Find]** Command

The **[Find]** command allows you to locate specific textual references in your ladder logic.

#### ■ To use the Find command:

1. If you have any windows open, close them before you use the **[Find]** command.
2. From the **[Search]** menu, select **[Find]**. The **[Find Text]** dialog box appears:



**Note:** The **[Find Text]** dialog box can also be opened by clicking  in the tool bar.

#### ◆ Specifying the type of matching to apply to the search

- You can specify the type of matching to apply to the search. If you were trying to find the word 'Fill', the Logic Program Editor would find all instances of that word, even if it found it as a lower case 'fill' or as part of another word such as 'Fillet'.
- If you selected **[Case sensitive]**, the Logic Program Editor would find 'Fill' but not 'fill'. If you selected **[Whole words only]**, the Logic Program Editor would find 'Fill' but not 'Fillet'.

#### ◆ Specifying the scope and direction of the search

- You can specify the scope and direction of the search. If **[Selection only]** is selected, the scope is limited to the highlighted portion of your program.
- Selecting **[Global]** includes the entire program. You can begin the search from the top of the selected scope by selecting **[Entire scope]** or from a given position by selecting **[From cursor]**. This tutorial starts the search from the beginning of the program.

## Chapter 1 – Creating a Program

3. Select the [START] label in your program.
4. Click the [Text to find] field of the [Field Text] dialog box.
5. Type 'FILL'.
6. Select [Global], [Forward], and [From cursor].
7. Click on the [Find] button. The “focus” moves to the first match found, a part of the 'Fill\_Timer' variable.
8. Click on the [Find] button again. The “focus” moves to the next match found. When you have reached a point in your program where there are no more instances of the items you are trying to locate, a beep sounds.



**Note:** After the first [Find] operation. You can locate subsequent occurrences of a text match by selecting [Find Next] from the [Search] menu.

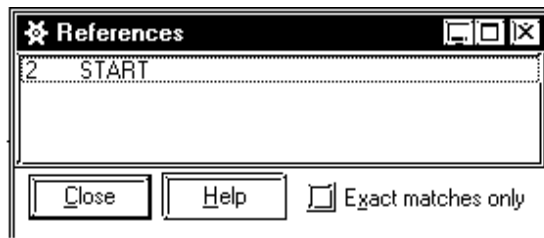
### 1.8.2 The [References] Command

The [References] command allows you to locate all occurrences of a specific variable in your ladder logic program. It identifies the rung numbers and the instructions the variable appears on.

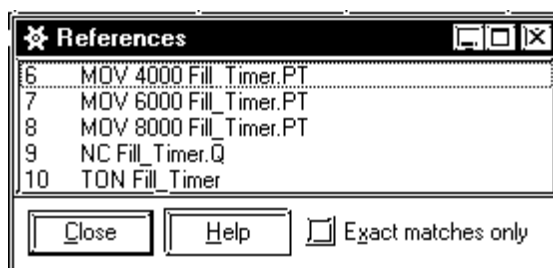
For this tutorial, you will select the [START] label. However, the [References] command can be implemented from any point in your program.

#### ■ To use the Reference command:

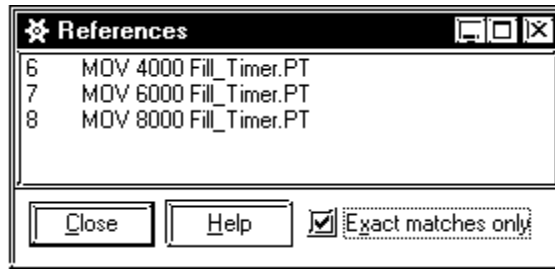
1. Click on the [START] label.
2. From the [Search] menu, select [References]. The [References] dialog box appears:



3. Re-size and move the [References] dialog box to the lower right hand corner of your screen.
4. Click on the rung 6's "Fill\_Timer.PT" variable and the [References] dialog box will appear as follows:



5. Select [**Exact matches only**].



In the [**References**] dialog box display:

- The number at the left of the line signifies the rung number the variable appears on. This display tells you the ‘Fill\_Timer’ variable appears on rung 6,7,8,9 and 10. When [**Exact matches only**] is selected, the display shows that ‘Fill\_Timer.PT’ occurs on rung 6,7 and 8.
- The next column on the line is the instruction type. This is the instruction that this variable has been assigned to on this rung. This display tells you the ‘Fill\_Timer’ variables has been referred by three (3) [**MOV**] instructions, one [**NC**] instruction and a [**TON**] instruction.
- The last column on the line lists the parameter that has been assigned to this instruction, including the variable you initially referenced. In this display, you can see the integers 4000, 6000 and 8000 assigned to the IN elements, and ‘Fill\_Timer.PT’ assigned to OUT elements.

The [**References**] dialog box changes in accordance with your selection every time you click on a variable in your ladder logic program. One advantage is when you click on any of the lines in its display, the corresponding point in your ladder logic appears.



**Note:**

**You must click on the parameter itself, not the instruction for the corresponding information to be displayed in the [**References**] dialog box.**

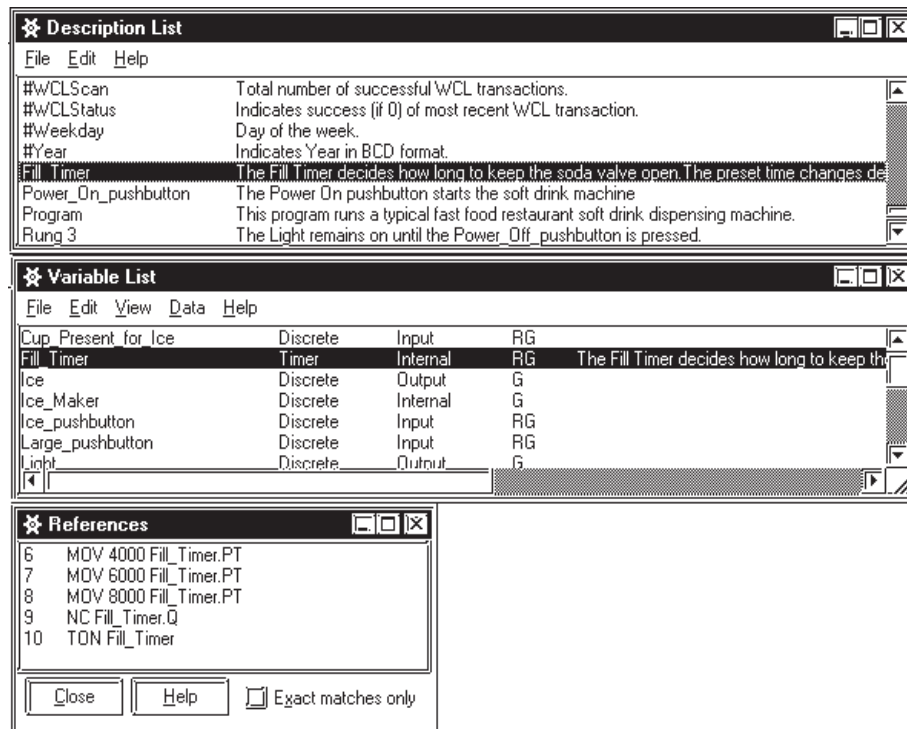


### 1.8.3 [References] Dialog Box with Other Dialog Boxes

Using only the [Reference] dialog box when you do not know where at least one instance of the desired variable is located is not the most convenient search method. You can also use the [Find] command to locate it, however, there is an even quicker method. You can use the [References] dialog box in conjunction with the [Variable List] and/or the [Description List] dialog box.

#### ■ To use the references dialog box with other dialog boxes.

1. Open the [Variable List], [Description list] and [References] dialog boxes.
2. Move and re-size them until your screen appears as follows:



3. Click on the variable 'Fill\_Timer' in the [Variable List] dialog box.



#### Note:

The displays of the [Description List] and [References] dialog box will change according to your selection. The [References] dialog box now displays every instance of the variable 'Fill\_Timer'. Also, note that even though you change a dialog box's display, the ladder logic program's display does not change. The corresponding point in your logic will appear when you select any variable line in the [References] dialog box.

4. Click on the first line in the [References] dialog box. Your ladder logic program now displays that variable highlighted on the rung and the instruction you specified.

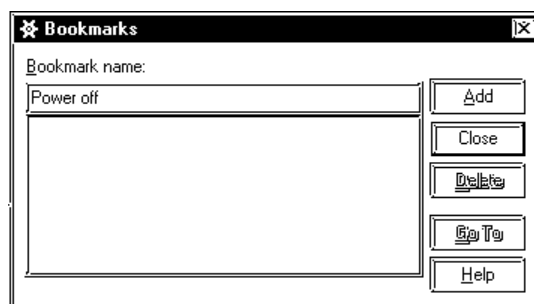
## 1.8.4 Using Bookmarks

If you are constantly referring back to a specific point in your ladder logic program, using a **[Bookmark]** saves you repeatedly scrolling the screen.

To set a **[Bookmark]**, you must signify the exact point where you wish to return to. Anything you can select or highlight can be a **[Bookmark]**. For this demonstration, the **[NORMALLY CLOSED CONTACT (NC)]** instruction on rung 3 is set as a **[Bookmark]**.

### ■ To set a [Bookmark]

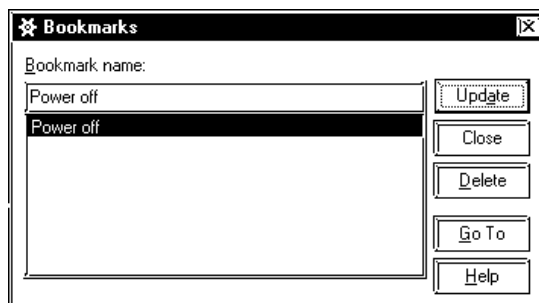
1. Click on the **[NC]** instruction on rung 3.
2. From the **[Search]** menu, select **[Bookmark]**. The **[Bookmarks]** dialog box appears.



3. Type **'Power Off'** in the **[Bookmark name]** field, then click on **[ADD]**. The **[Bookmark]** has now been set. Thus, whenever you select **'Power Off'** and click on **[Go To]** to return to your **[Bookmark]**, you will return to the **[NC]** instruction on rung 3. If you wish to set a new **[Bookmark]**, simply select a new point on the ladder logic and repeat steps 1 through 3. The Logic Program Editor supports the use of multiple **[Bookmarks]**.

### ■ To go to a [Bookmark]

1. From the **[Search]** menu, select **[Bookmarks]**. The **[Bookmarks]** dialog box appears.



2. Select a **[Bookmark Name]** from the list, then click on **[Go To]**. Wherever you are in your ladder logic program, the LT Editor automatically takes you back to where you placed the **[Bookmark]**.



**Note:** You can use the **[CTRL] + [M]** keys to open the **[Bookmarks]** dialog box.

### ■ To change the position of a [Bookmark]

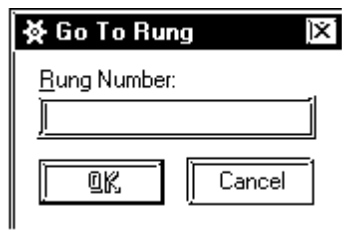
1. Select the new position in the ladder logic program.
2. Select the [Bookmark name] you wish to reposition.
3. Click on [Update] in the [Bookmarks] dialog box.

### 1.8.5 Using the [Go To Rung] Command

The [Go To Rung] command allows you to move the “focus” to a specified rung in your ladder logic program.

### ■ To use the [Go to Rung] command

1. From the [Search] menu, select [Go To Rung] and the following dialog box will appear:



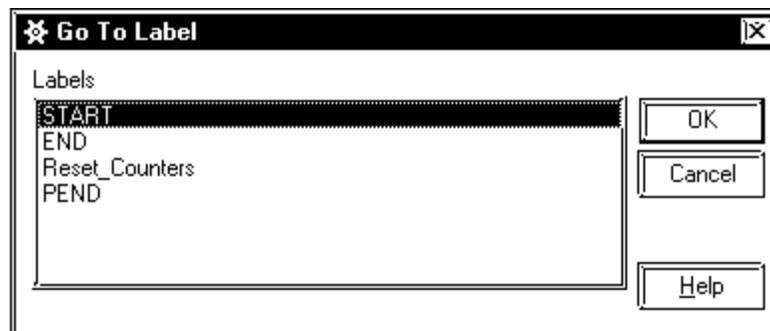
2. Enter a [Rung Number].
3. Click on [OK]. You are now positioned at the specified rung.

### 1.8.6 Using the [Go To Label] Command

The [Go to Label] command allows you to jump to a specific “label” in your ladder logic program.

### ■ To use the [Go to Label] command:

1. From the [Search] menu, select [Go TO Label]. The [Go To Label] dialog box appears:



2. Select the label to go to.
3. Click on [OK]. You are now positioned at the specified label.

## Summary

This section has explained how to use [Find], [References], [Bookmark], [Go To Rung] and [Go To Label] commands.

## 1.9 I/O Configuration

Once you have finished constructing a ladder logic program, you must assign I/O to selected variables. In this tutorial, variables were created first and I/O assigned after the ladder logic program was completed. This was done in order to present the various features of the LT Editor in a logical order. If you know what your I/O will be before beginning programming, you can specify your I/O first and then assign it to your variables as you create your program. Both methods are demonstrated in this section.

### 1.9.1 Assigning Variables to I/O

Once you have created variables in a ladder logic program, there are a number of methods you can use to assign them to your I/O.

The "Ice\_pushbutton", "Large\_pushbutton", "Medium\_pushbutton", and "Small\_pushbutton" will be placed on the LT screen for touch-panel inputs. These buttons are not assigned to the terminals.

Variable Name	Terminal Type	Terminal No.
Power_ON_pushbutton	Input	I0
Cup_Present_for_Ice	Input	I2
Power_OFF_pushbutton	Input	I6
Light	Output	Q0
Ice	Output	Q1
Soda_valve	Output	Q2

Due to the differences of the I/O drivers, the procedures for “Opening the [Configure I/O] window” and “Setting up the driver” on the LT Type A differ from the procedures used for LT Type B, LT Type B+, and LT Type C.

**Reference** *When using a Type A unit, please refer to “To set up the DIO driver”.*

*When using Type B, Type B+, and Type C units, please refer to “To set up the Flex Network driver”.*

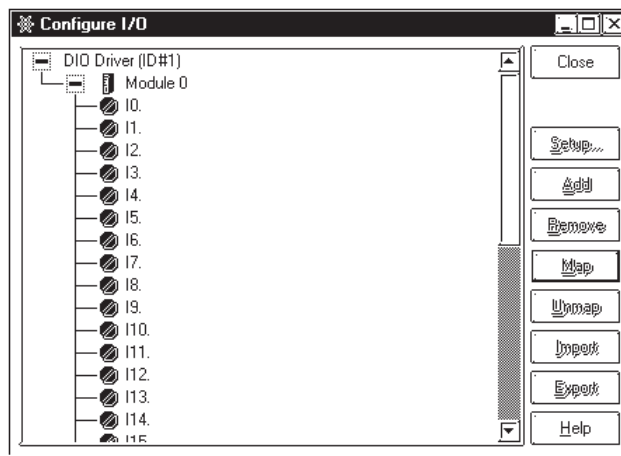
*When using an LT Type H unit, please refer to the LT Type H driver manual (sold separately).*

The explanations in the tutorial lessons so far used the LT Type A as the model environment. However, the “To set up the Flex Network driver” section uses the LT Type B+ as the model environment.

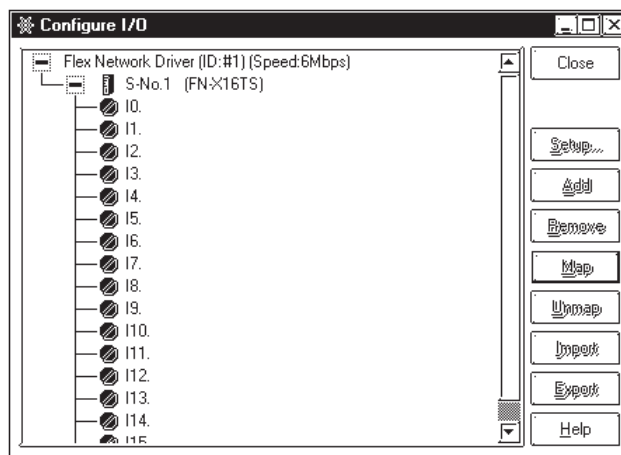
### ■ To open the [Configure I/O] window:

From the [Data] menu, choose [Configure I/O] and the following window will appear.

#### ◆ For the DIO driver



#### ◆ For the Flex Network driver

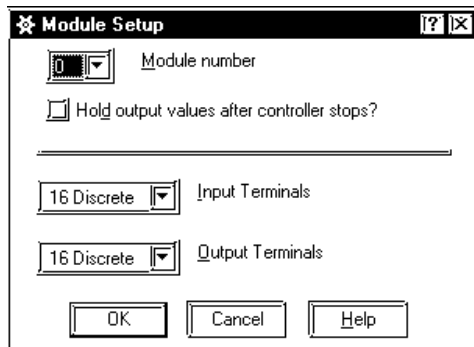


**Note:**

You can also open the [Configure I/O] dialog box by clicking on  on the tool bar or by clicking on  in the [Variable Type] dialog box.

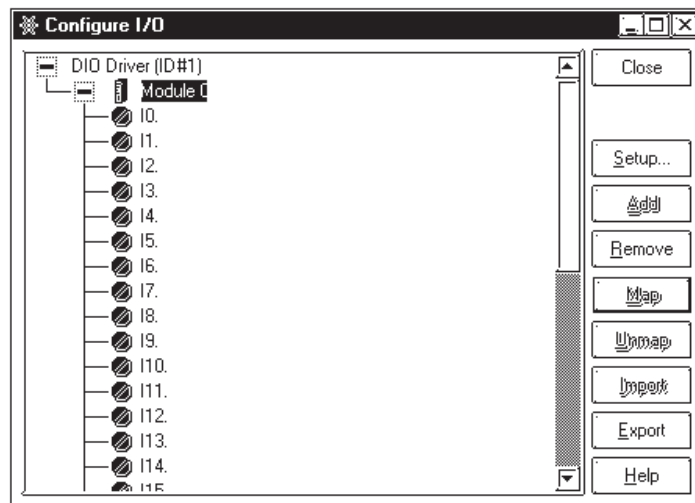
■ **To set up the DIO driver:**

1. Select 'Module 0'.
2. Click on [Setup]. The [Module Setup] dialog box appears:



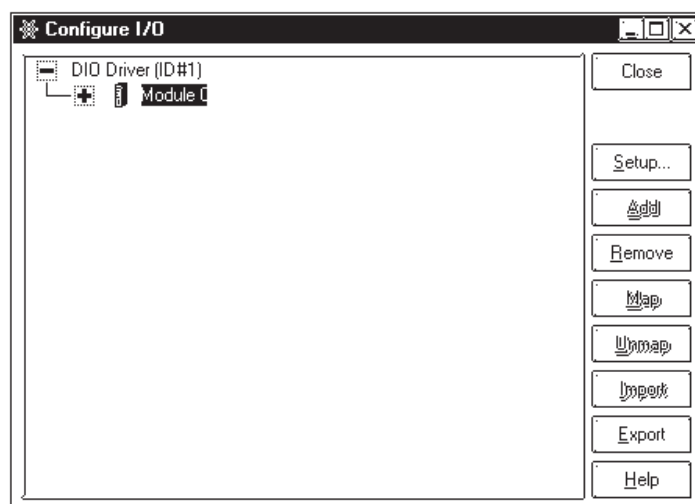
\* Only "0" can be designated for a Module number.

3. "16 Discrete (bit)" is factory set for both Input/Output terminals.
4. Click on [OK]. The [Configure I/O] window appears as follows:



Displayed underneath Module 0 are 16 input terminals and one output terminal (for a 16-bit word) associated with the DIO module displayed. You will assign variables to them later in this tutorial.

5. Click on  next to Module 0. The terminals are hidden and  appears in place of .



■ **To set up the Flex Network driver:**

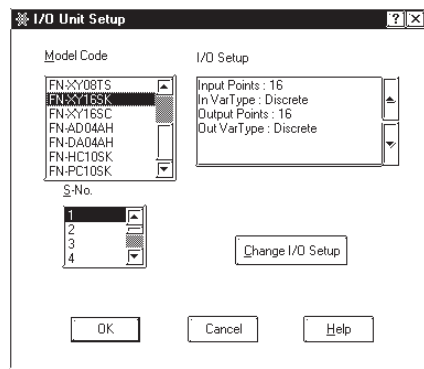
When assigning the B+ unit’s built-in I/O, setup the following driver first. (The DIO built into the LT Type-B+ is treated as a single Flex Network station.)

Model Code: FN-XY16SK

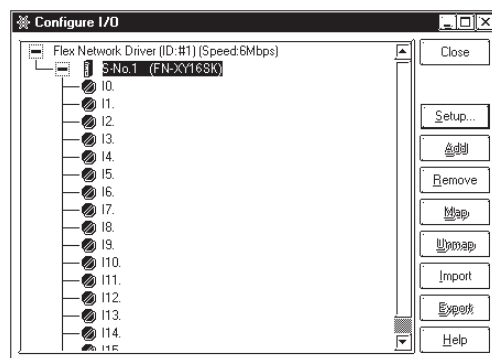
S-No.: Select a number that will not overlap with other connected devices.

In this lesson, select “1”.




1. Select "**S-No. 1 (FN-XY16SK)**" in the Configure I/O window, then click [**Setup**]. The [**I/O Unit Setup**] dialog box appears.

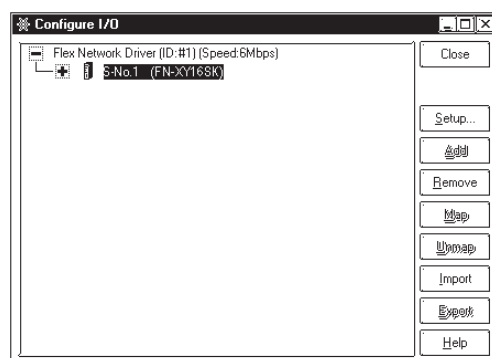


2. Change "**FN-X16TS**" to "**FN-XY16SK**" in the “Model Code” field.
3. Click on [**OK**]. The [**Configure I/O**] window appears as follows:



Displayed underneath ‘S-No.1 (FN-XY16SK)’ are 16 input terminals and one output terminal (for a 16-bit word) associated with the Flex Network module displayed. You will assign variables to them later in this tutorial.

5. Click on  next to ‘S-No.1 (FN-XY16SK)’. The terminals are hidden and  appears in place of .

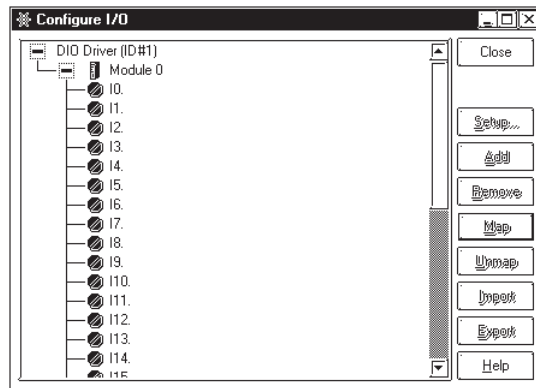


6. Up to 63 units (when 2 lines are used) can be connected with the Flex Network driver. Use the same method for selecting a module for another unit.


■ To click and drag variables to the I/O terminals:

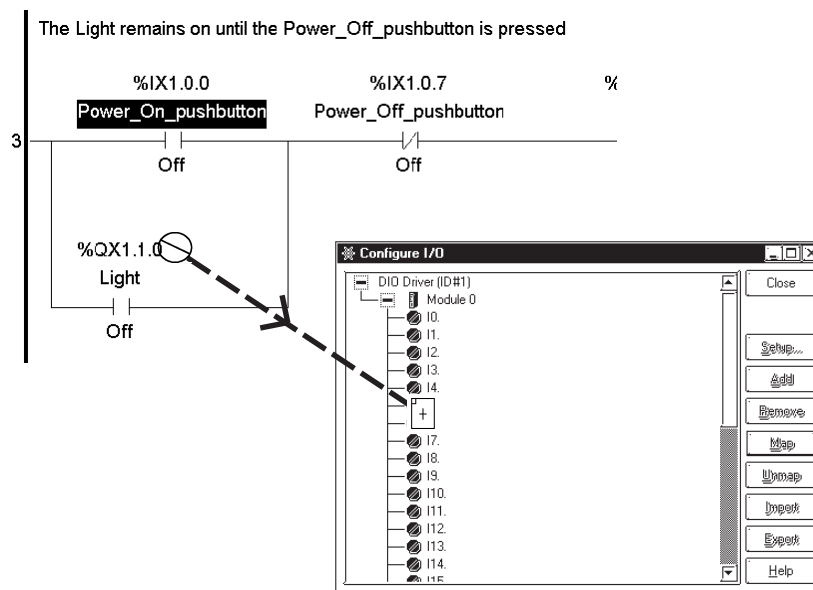
1. Click on  next to 'Module 0'. The [Configure I/O] window appears as follows:

You can use the first 16 terminals for input with Module 0.

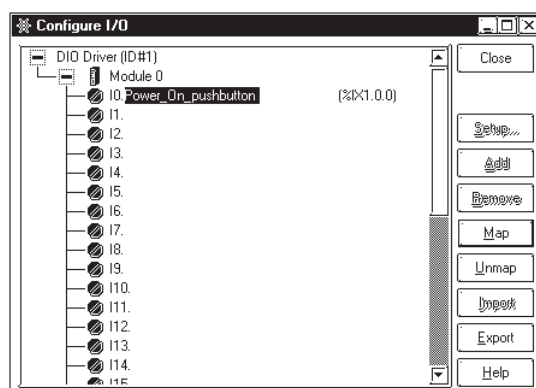


2. Locate the variable 'Power\_On\_pushbutton' on the NO instruction of rung 3.

3. Click and drag 'Power\_On\_pushbutton' toward terminal I0. As well as when inserting branches, note that your cursor initially becomes a . When the cursor is in this state you cannot assign the variable to any I/O terminal.



4. Drag the cursor over terminal 0 and release the mouse. The variable 'Power\_On\_pushbutton' is now assigned to terminal I0.

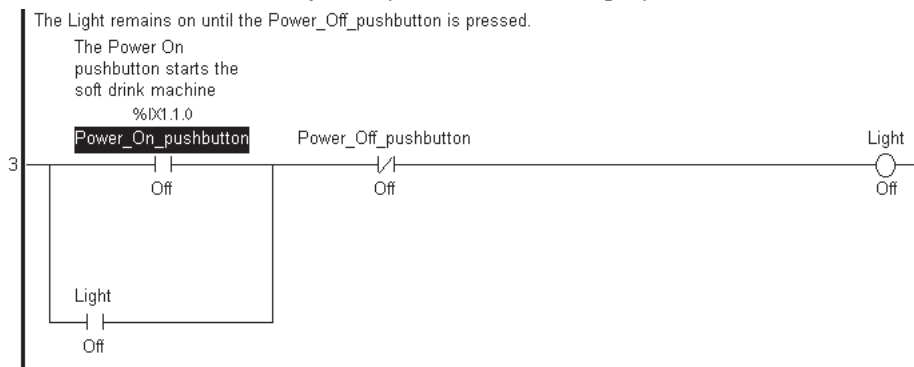




## Chapter 1 – Creating a Program

The variable 'Power\_On\_pushbutton' on the NO instruction of rung 3 now has a series of digits and letters above it. This is the IEC I/O address of that variable.

**Reference** For more information about the IEC addressing format of your I/O driver, refer to your driver's Help system.



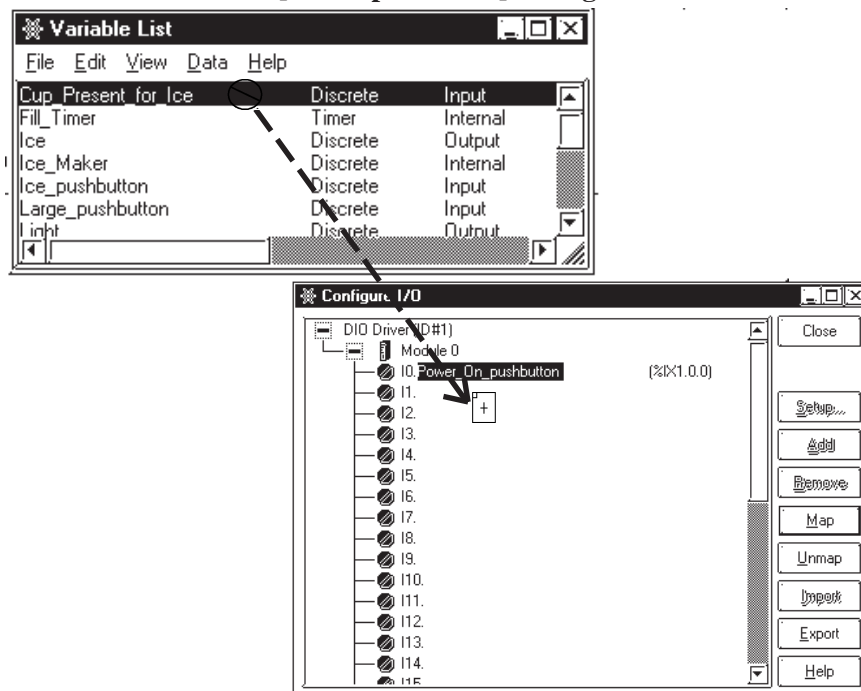
### ■ To click and drag variables to I/O terminals from the [Variable List] dialog box:

1. Open the [Variable List] dialog box. The [Configure I/O] window should still be open.
2. Arrange the dialog boxes so that both can be viewed.
3. From the [Variable List] dialog box, click and drag the variable 'Cup\_Present\_for\_Ice' to terminal I2 in the [Configure I/O] window.
4. Release the mouse. The variable 'Cup\_Present\_for\_Ice' is now assigned to input terminal I2.



**Note:**

You can also use the above procedure to assign variables to I/O from the [Description List] dialog box.

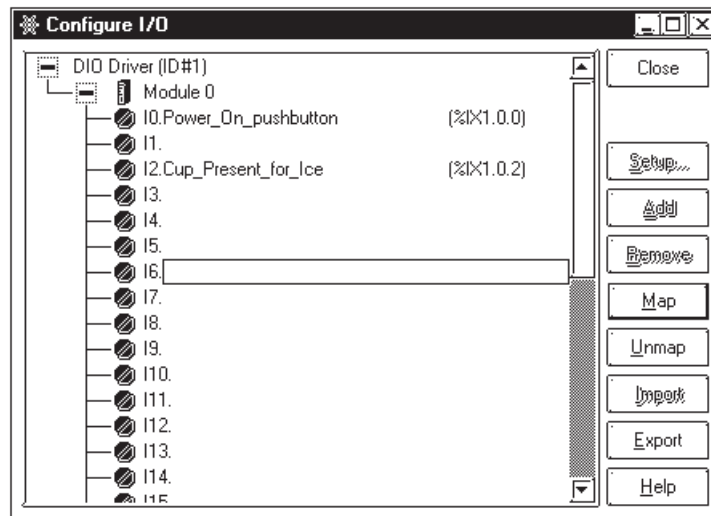


**Important**

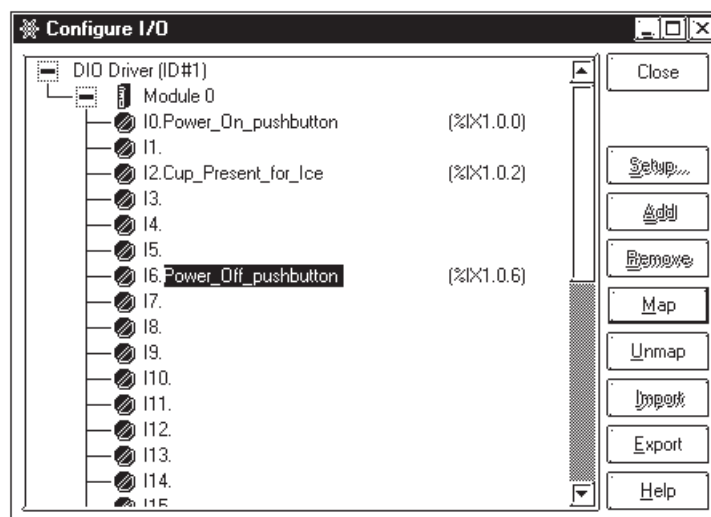
When you assign (click and drag) a variable to [Configure I/O] from the [Variable List] or [Description List] window, that I/O attribute is enabled and any other variable attribute will be changed to Input/Output.

■ **To assign variables via text input:**

1. Click on terminal I6.
2. Press the [Enter] key. The terminal test field is activated.



3. Type 'Power-Off-pushbutton'.
4. Press the [Enter] key. 'Power-Off-pushbutton' is now assigned to input terminal I6.



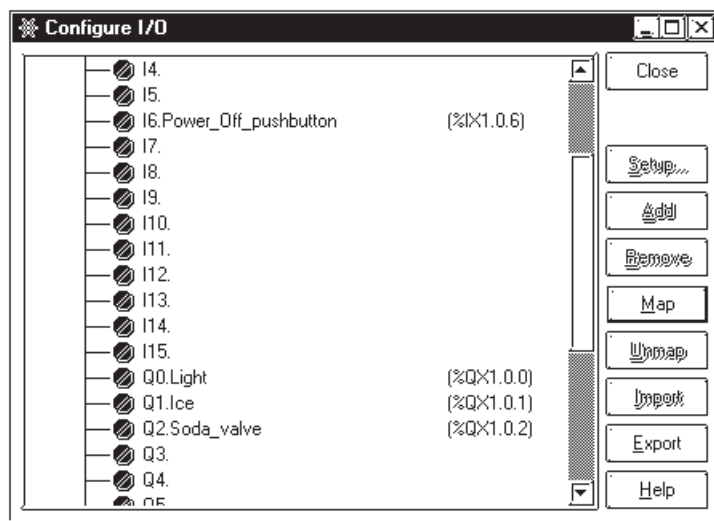
**Note:** When variables are assigned to I/O via text entry, the variables will be automatically listed in the [Variable List] dialog box.

## Chapter 1 – Creating a Program

Assigning variables to output terminals is the same as assigning them to input terminals. Use the above procedures to assign variables from the following table to the input and output terminals.

Variable Name	Terminal Type	Terminal #
Light	Output	Q0
Ice	Output	Q1
Soda_valve	Output	Q2

The input and output modules are displayed in the [Configure I/O] dialog box as shown here:



### 1.9.2 Unassigning Variables from the [Configure I/O] Dialog Box

#### ■ To unassign a variable from the [Configure I/O] window:

1. Click on terminal I0 in the [Configure I/O] window.
2. Click on [Unmap]. The 'Power\_On\_pushbutton' is now unassigned from terminal I0 and can be assigned to any other terminal you select. In this tutorial, assign it back to terminal I0.

### 1.9.3 Assigning I/O to Variables

The easiest way to configure I/O for new programs is to type the variables directly into the terminals. They are then automatically created, configured and mapped to the correct I/O point. In this case, when you configure your I/O first and then construct your ladder logic program, creating your I/O points is explained.

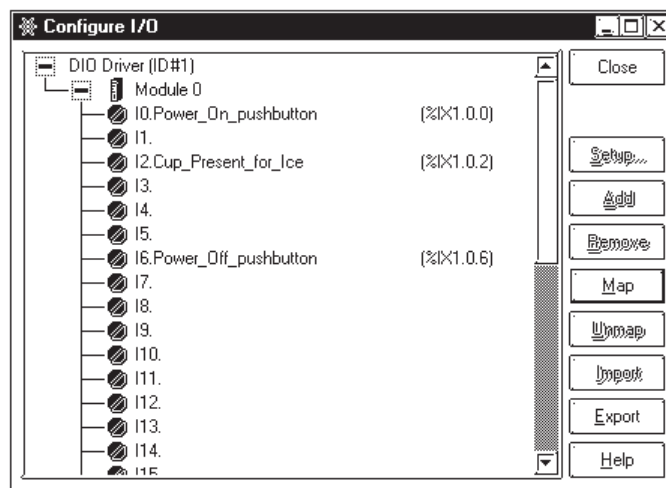
#### ■ To use variables assigned to I/O with Instructions:

1. Click the target variable and drag to the I/O terminals as described above to assign variables to the input and output terminals of your driver.
2. Construct your ladder logic program.
3. Click and drag the variables from the [**Configure I/O**] dialog box to the instructions you want I/O assigned to.

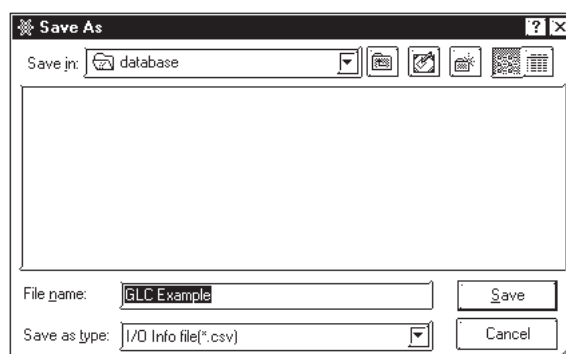
### 1.9.4 Converting I/O Configuration Data

The variables assigned to the I/O of the LT Type A are automatically converted to the Flex Network. The variables can be converted to an FN-X32TS 32-point I/O unit, FN-XY16SK, FN-XY16SC, or the I/O (FN-XY16SK) that are built in the Type B+ unit.

This section describes the steps used to convert LT Type A I/O to LT Type B+ I/O (FN-XY16SK).



1. Click the [**Export**] button on the [**Configure I/O**] window. Save the variables assigned to the I/O of the LT Type A in a CSV format file.

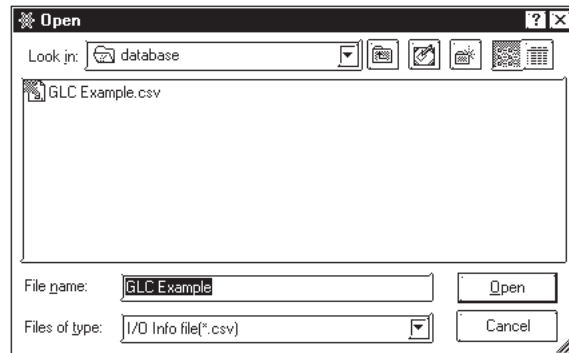


## Chapter 1 – Creating a Program

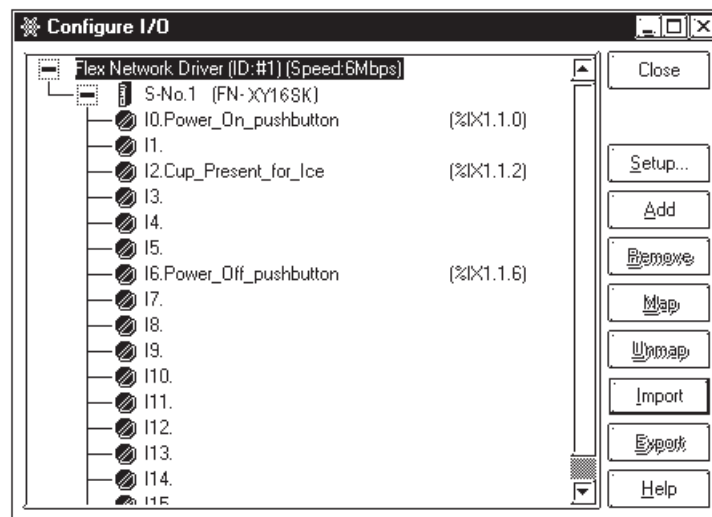
2. In the [New/LT Type] in the Project Manager, change the model type from “Type A” to “Type B+”.

### **Reference** 1.1 How to Start the LT Editor

3. Open the [Configure I/O] window via the Logic Program Editor.  
Select [S-No.1 (FN-XY16SK)] and then click the [Import] button. Select the previously saved CSV file and click on [Open].



4. The variables are imported from the CSV file and assigned to the LT Type B+ unit's I/O (FN-XY16SK).



## Summary

This section explained how to:

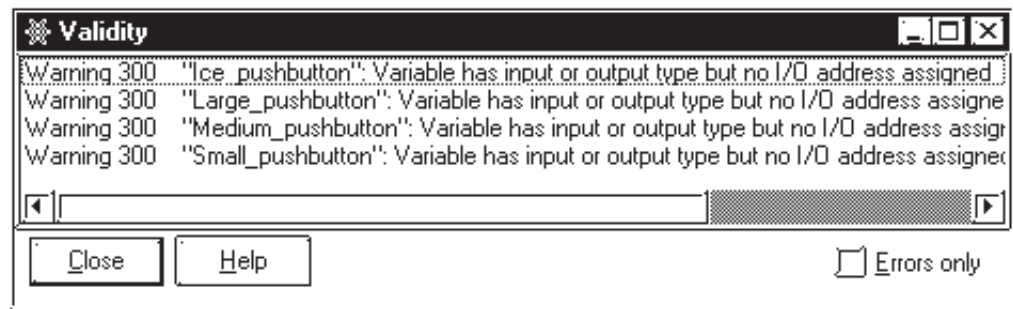
- select an I/O driver,
- configure the Flex Network driver,
- use variables assigned to I/O.

## 1.10 Checking the Validity of a Program

Before running a ladder logic program online, use a validity check to make sure the program is free of errors.

### ■ To run a validity check

From the [File] menu, select [Check Validity] and the following dialog box will appear.



The [Validity] dialog box lists all errors and possible trouble spots the Logic Program Editor can detect in your program. Trouble spots are listed as “warnings”.

In the lower right hand corner of the dialog box is a check box marked [Errors only]. If this box is selected, only the “errors” that the Logic Program Editor detects in your program are displayed; the “warnings” are not. The Logic Program Editor can run a program that contains “warnings” in the Controller, however, it cannot run a program that contains errors. These errors must be corrected first.



**Note:** A validity check can also be performed by clicking on  in the tool bar.

The [Validity] dialog box displays “errors” and “warnings” in the order they appear in your ladder logic program. In other words, the “errors” in rung 1 are presented first, then rung 2 and so on.

If you double-click on the “errors” or “warnings” in the [Validity] dialog box you can go directly to the problem.

- If it is a logic problem, that part of your program is displayed.
- If it is a problem with assigning I/O, the [Configure I/O] dialog box is displayed.

As previously mentioned, there can be a variety of “error” types displayed in the [Validity] dialog box. Your validity check will show the following error.

Error 200      Rung 9: Parameter should be a Discrete

### ■ To fix an error

1. Double-click on the “error” line in the [Validity] dialog box. The [Instruction Parameter Box] of the instruction on rung 9 is highlighted, indicating there is no variable assigned to it.
2. Enter “Soda\_valve” as the instruction variable.

**Reference** *For more information on specific errors and warnings, refer to the Editor Help system or “Chapter 4 Errors and Warnings” in this manual.*

When you have corrected the “errors” listed in the [Validity] dialog box, run a validity check again. Any errors that exist are displayed. If they have all been corrected your program can be written to the Controller.

### Summary

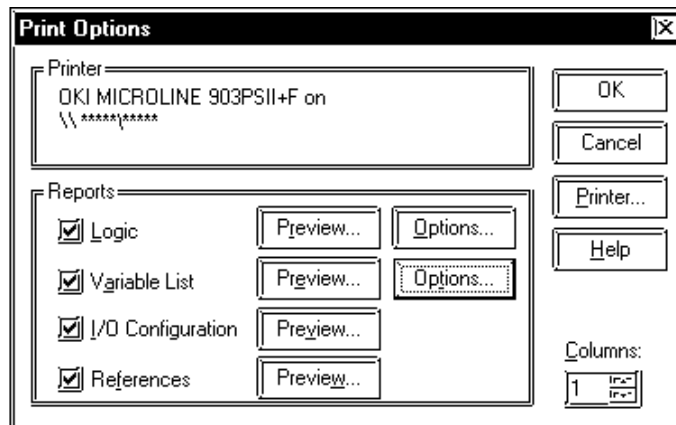
In this section you have learned how to check the validity of an Editor ladder logic program. The preparation for transferring a program to the LT for execution is complete. The details of the procedures hereafter are explained in 2.1 Configuring the LT Controller.

## 1.11 Printing Your Ladder Logic Program

With the Logic Program Editor, you can print different aspects of your ladder logic program.

### ■ To print a ladder logic program

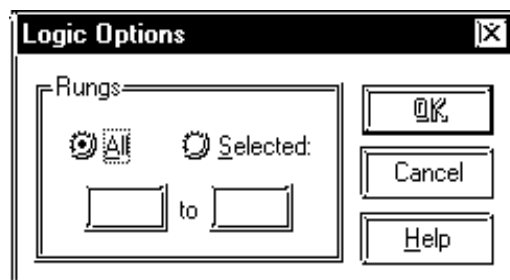
From the [File] menu, select [Print] and the following dialog box is displayed. You can view the logic program on the screen before it is printed using the Preview function.



You can select the number of columns (1 to 4) into which your report will be formatted. Under the [Reports] section there are four check boxes labelled [Logic], [Variable List], [I/O Configuration] and [References]. These check boxes provide the following options when printing your ladder logic program:

#### ◆ [Logic]:

This option allows you to print the rungs of your ladder logic program. If you click on [Options] next to it, the following dialog box appears:



Select [All] to print all the rungs of the program or, click on [Selected] and type in the range of rungs you wish to print.

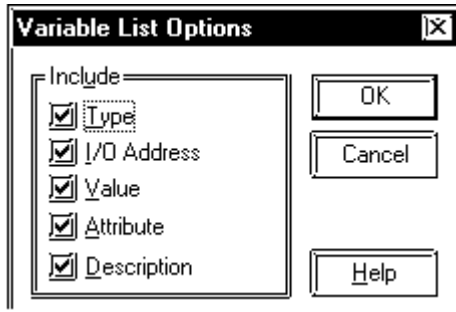
Use the [View] menu to adjust the logic program's printout size.



## Chapter 1 – Creating a Program

### ◆ [Variable List]

This option allows you to print a variable list. Click on [Options] to select the items you wish to include in that variable list.



Option	Description
Type	Displays the variable type.
I/O Address	Displays the I/O addresses of all assigned variables.
Value	Displays the data value of all variables.
Attribute	Displays the Retentive and Global settings
Descriptions	Displays any descriptions given to the variables.

### ◆ [I/O Configuration]

This option allows you to print your I/O configuration.

### ◆ [References]

This option allows you to print a cross reference report showing all instances of all variables.



**Note:** You can also print your program by clicking on  in the tool bar.

## Summary

This section explained how to select which aspects of your ladder logic program you wish to print.

## 1.12 Importing/Exporting a Logic Program

The Logic Program Editor allows you to export a logic program exclusively for use with the LT.

Conversely, a logic program file can be imported for use as a project file for another LT. You can import and export all logic programs created in the project. You can also export part of a logic program using [**Export** | **Part**], and import part of a logic program using [**Import** | **Insert**].

### 1.12.1 Export

The following three types of logic programs can be exported.

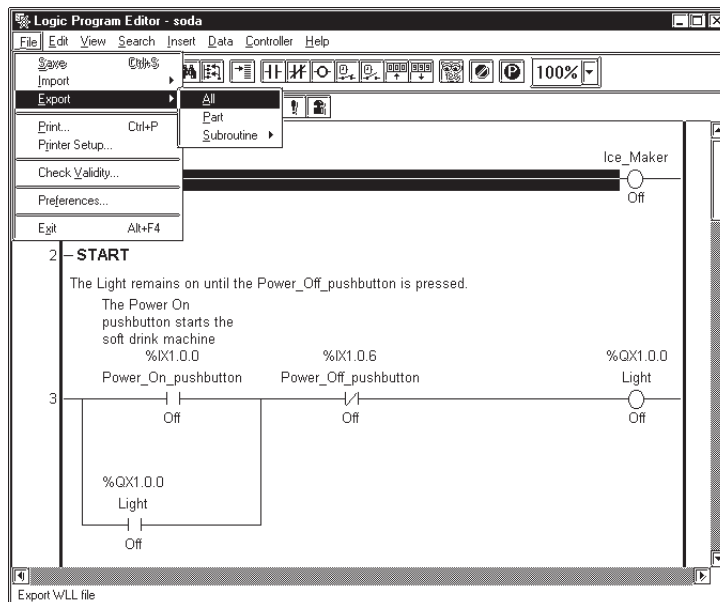
- All logic programs including subroutines (\*.wll)
- A selected part of a logic program (\*.wlp)
- Subroutine in a logic program (\*.wlf)

#### ■ To Export a Logic Program

Procedures for exporting the above three types of logic programs are explained as follows.

#### ◆ To Export All Logic Programs including Subroutines

1. Select the [**Export** | **All**] command from the [**File**] menu.



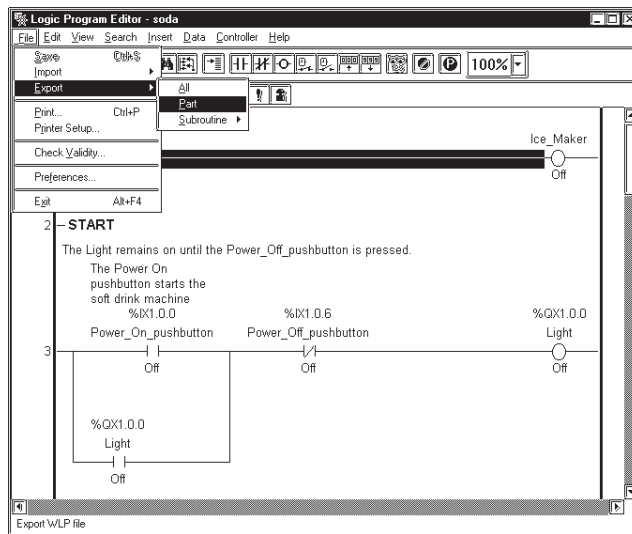
2. Enter a file name in the [**Save As**] dialog box.
3. Click [**Save**].

The Logic Program is saved in .WLL format.

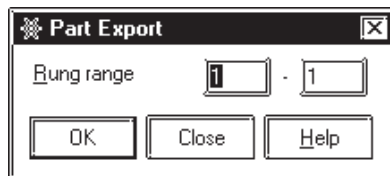
## Chapter 1 – Creating a Program

### ◆ To Export Selected Part of a Logic Program

1. Select the **[Export | Part]** command from the **[File]** menu.



2. Select the rungs to be exported and click **[OK]**.

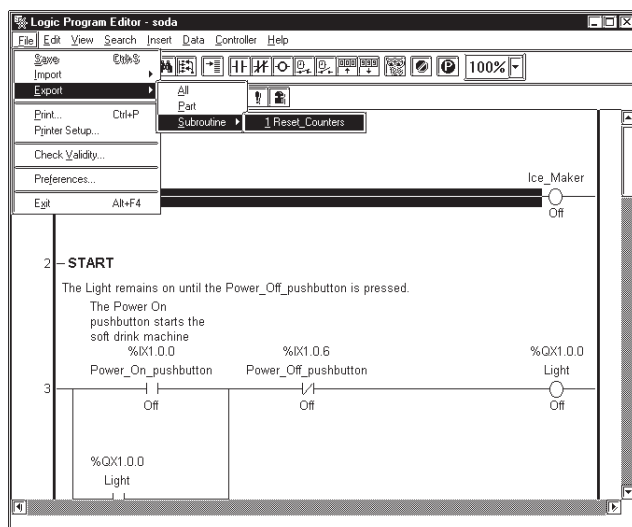


3. Enter a file name in the **[Save As]** dialog box.
4. Click **[Save]**.

The Logic Program is saved in .WLP format.

### ◆ To Export Subroutine in a Logic Program

1. Select the **[Export | Subroutine]** command from the **[File]** menu. When **[Subroutine]** is selected, a list of subroutines created in the logic program is displayed. Select the subroutine to be exported from the list.



2. Enter a file name in the **[Save As]** dialog box.
3. Click **[Save]**.

The Logic Program is saved in .WLF format.

## 1.12.2 Import

The following three import commands can be used to import logic programs.

- [Update] command – imports all logic programs including subroutines
- [Insert] command – imports a selected part of a logic program
- [Subroutine] command – imports a subroutine part

Please note that when importing all logic programs, including subroutines, the logic program is updated to a logic program in the current project.

The location where imported rungs are inserted can be set up with the [**File** | **Preferences** | **Editor**] command.

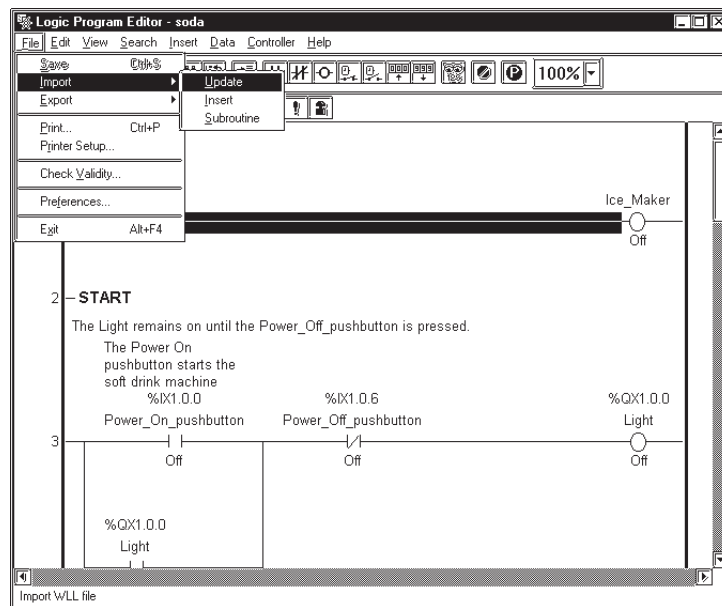
**Reference** Please see the “Preference Area Settings (Prior to Creating a Logic Program)” section at the beginning of Chapter 1.

### ■ To Import a Logic Program

Procedures for importing logic programs using the above three methods are explained as follows.

#### ◆ To Import All Logic Programs including Subroutines

1. Select the [**Import** | **Update**] command from the [**File**] menu.



2. Select the .WLL file you want to import in the [**Open**] dialog box.
3. Click [**Open**].

The specified logic program is imported, and the variables used in the logic program are registered to the Variable List.

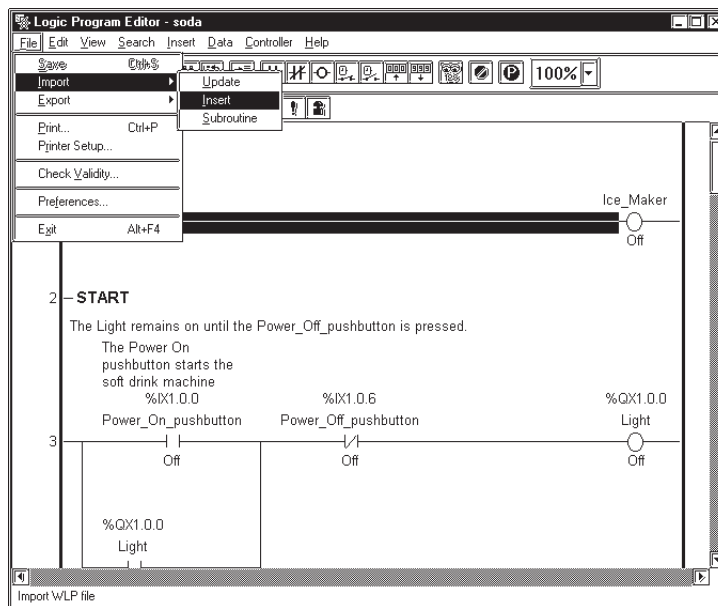
4. Saving the logic program will register a global variable in the Symbol Editor as a logic symbol.

**Reference** *Operation Manual – Screen Creation Guide, 4.2.5 – “Symbol Editor.”*

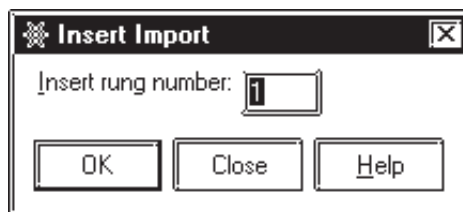
## Chapter 1 – Creating a Program

### ◆ To Import Selected Part of a Logic Program

1. Select the [Import | Insert] command from the [File] menu.



2. Specify a location (rung number) to insert the logic program.



3. Select the .WLP file you want to import in the [Open] dialog box.
3. Click [Open].

The specified logic program is imported, and the variables used in the logic program are registered to the Variable List.

4. Saving the logic program will register a global variable in the Symbol Editor as a logic symbol.

**Reference** *Operation Manual – Screen Creation Guide, 4.2.5 – “Symbol Editor.”*



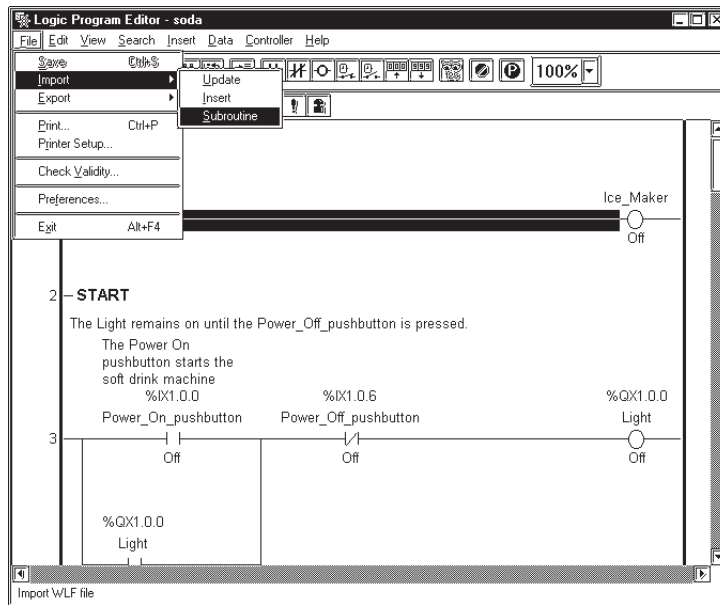
**When the imported logic program contains variables with the same name as variables in the current logic program, the imported logic program’s variable types are changed to match those of the current logic program.**

### Summary

In this section, you have learned how to import and export a logic program.

## ◆ To Import Subroutines of a Logic Program

1. Select the **[Import | Subroutine]** command from the **[File]** menu.



2. Select the .WLF file you want to import in the **[Open]** dialog box.
3. Click **[Open]**.

The specified logic program is imported, and the variables used in the logic program are registered to the Variable List.

4. Saving the logic program will register a global variable in the Symbol Editor as a logic symbol.

**Reference** *Operation Manual – Screen Creation Guide, 4.2.5 – “Symbol Editor.”*

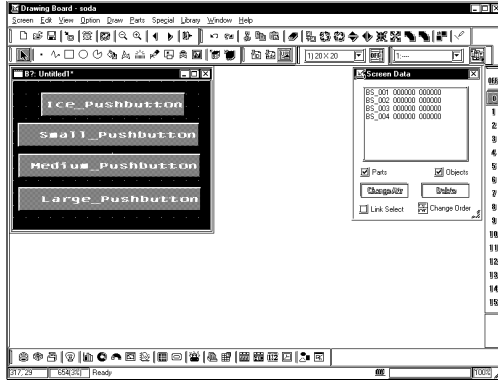


**Important**

**When the existing logic program contains variables with the same name as variables in the current logic program, the imported logic program’s variable types are changed to match those of the current logic program.**

## 1.13 Developing a Screen Program

Create “Ice\_Pushbutton,” “Large\_Pushbutton,” “Medium\_Pushbutton,” and “Small\_Pushbutton” with the Drawing Board (Screen Editor). The illustration below is the completed sample screen.

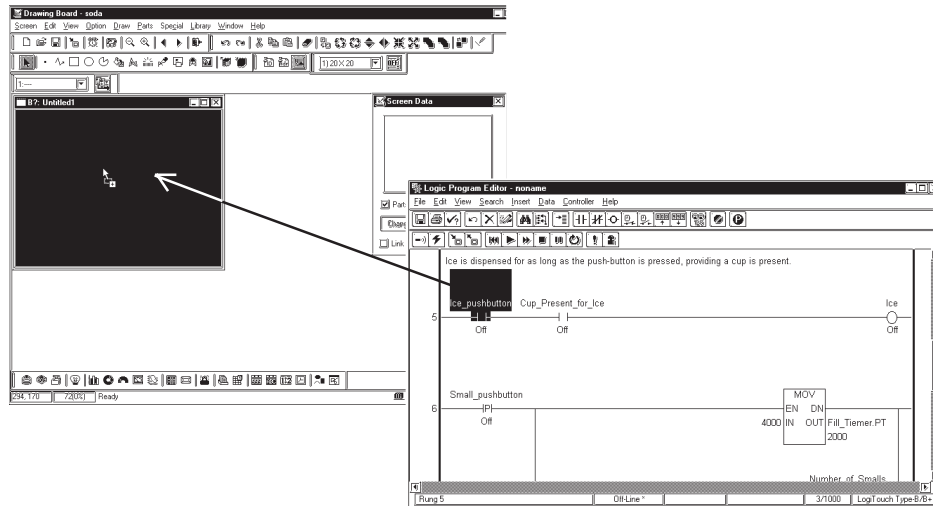


### ■ To Start the Drawing Board

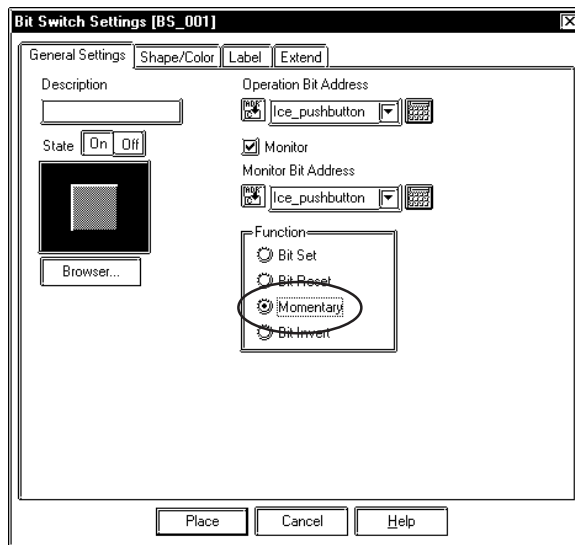
1. In the Project Manager window, click [**Draw | Screen**] to activate the Drawing Board (Screen Editor).
2. Click [**Screen | New**] on the Menu Bar. Check that "Base Screen" is selected, and click the [**OK**] button.

### ■ To Draw using Drag and Drop Operations

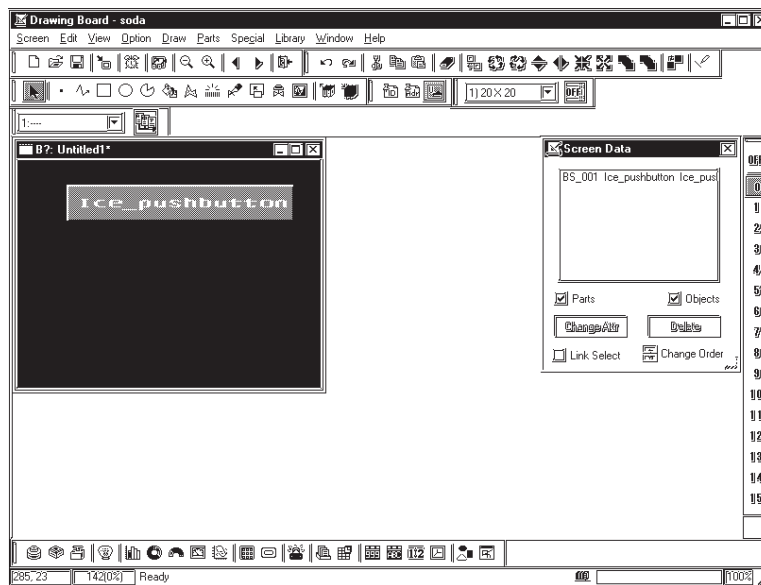
1. Select “Ice-pushbutton” in the Logic Program, and drag it to the Drawing Board (Screen Editor).



2. The [**Bit Switch Settings**] dialog box appears on the screen. Select "Momentary" from the [**Function**] field. Check that the [**Operation Bit Address**] is set to [**Add Ice**], and then click the [**Place**] button to place the pushbutton.



3. "Add Ice" is completed. Create "Large\_pushbutton," "Medium\_pushbutton," and "Small\_pushbutton" using the same procedure.





# *Memo*

## 2 Running the Ladder Logic Program

Once you have developed a ladder logic program that is free of errors, it can be run by the LT Controller.

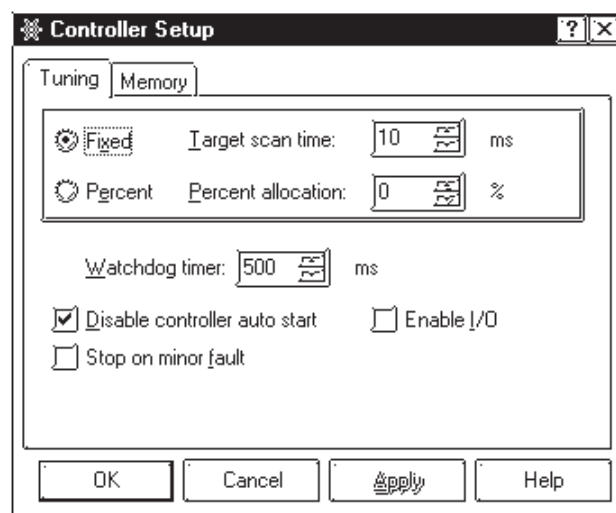
This chapter explains how to configure the LT Controller, send (write) a program to it and run the program online.

### 2.1 Configuring the LT Controller

Before writing a ladder logic program to a LT Controller, please be sure that the controller is configured properly. When running a program on the LT, there are two settings in the Controller Setup: “Tuning” and “Memory”.

#### ■ To Configure the Controller:

From the [Controller] menu, select [Setup], which calls up the following screen.



When you set parameters on the [Tuning] tab, you are setting the parameters the ladder logic program uses when it is written to the LT Controller. From this point onward, whenever this particular program is run, the LT Controller uses these settings, unless they are changed. These settings are unique to this program.

Controller [Tuning] options are explained below.

## Chapter 2 – Running the Ladder Logic Program

Option	Description
Target Scan Time	In [Target Scan Time] (System Variable: "#TargetScan"), enter the amount of time in milliseconds you would like each scan of your program to take. (Note): If the logic time exceeds the 50% of the scan time, the "Scan" operation is not guaranteed. Specify the setting in 10-ms increments.
Percent Allocation	In [Percent allocation] (System Variable: "#PercentAlloc"), enter a value in % to designate the scan time by the percentage of the whole dealing time. The 1-ms place of the calculated scan time is round up.
Watchdog Timer	When a logic program alarm occurs that delays the scan so that the value entered here is exceeded, a Major Fault alert occurs.
Disable Controller Auto Start	Only when the LT OFFLINE mode's "MODE WHEN POWER IS ON" selection is set to [DEFAULT] is this feature enabled. <sup>1</sup> When the controller is restarted after being stopped, this feature will automatically prevent the Logic Program from restarting. The system variable #DisableAutoStart can also be used for this setting. <b>Reference</b> 8.2.20 #DisableautoStart
Stop on Minor Fault	This setting designates if the logic program is stopped when a minor controller fault occurs. The system variable #FaultOnMinor can also be used for this setting. <b>Reference</b> 8.2.22 #FaultOnMinor
Enable I/O	This function enables the inputs/outputs to the LT main unit and external I/O of the I/O unit. In normal operation, the input/output of the external I/O is disabled when the LT is set to RUN mode after performing a Logic Program download. For safety reasons, this function prevents the possibility of accidental startups of machines caused by errors in operation and logic programs.

1. To set up the "MODE WHEN POWER IS ON", select [PLC Setup] - [Controller] - [Setup]. If [Start/Stop] is selected in the [Controller] menu, the settings of the LT Editor are ignored while the off-line settings are prioritized.

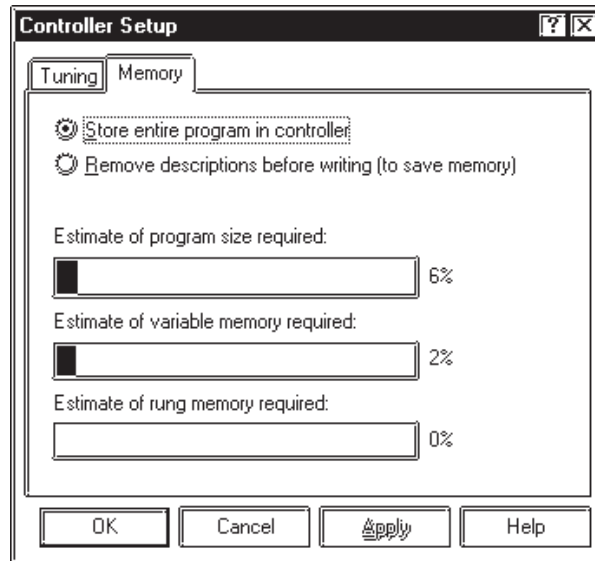


### Note:

- For details on Target Scan Time and Percent Allocation, refer to Chapter 6: Controller Features.
- For details on the system variables, refer to Chapter 8: System Variables.
- The "Enable I/O" feature can be selected when starting and stopping the controller. For details, refer to 2.2 Starting and Stopping the Controller.

### ◆ Memory

The [Memory] tab shows the percentages of [Estimate of controller memory required] and [Estimate of variable memory required] with bar graphs.



#### [Store entire program in controller]

Transmits the entire ladder logic program, including comments. Comments for the ladder logic program can be read when reading is done from the LT.

#### [Remove descriptions before writing (to save memory)]

Reduces the size of the file you are downloading to the LT, therefore, when the file is uploaded from the LT, there will not be any description data.

#### [Estimate of controller memory required]

Shows the memory of the current program as a percentage of the LT's usable memory.

#### [Estimate of variable memory required]

Shows the total memory of all variables currently registered as a percentage of the LT's usable memory.

#### [Estimate of rung memory required]

Shows the total memory of currently used instructions and rungs as a percentage of the LT's usable memory.

### 2.1.1 Writing to the Controller

After you have completed creating a ladder logic program with the Logic Program Editor and it is free of errors, you can write it to the LT and run it online.

To write a logic program to an LT, you can either;

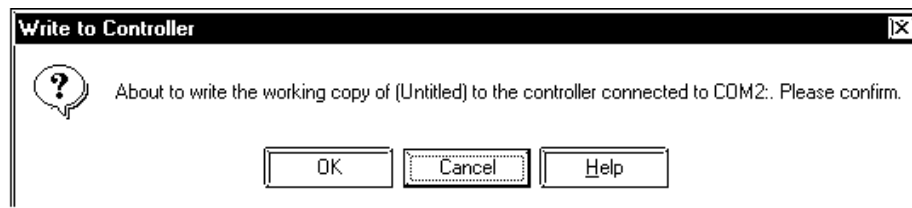
- transfer the screen data and logic program via the "Transfer" window of the LT Editor.
- transfer the logic program exclusively via the "Transfer" window of the LT Editor.

Make sure to set up your LT before writing the logic program. To set up an LT, transfer the system along with a Project File via the "Transfer" window of the LT Editor. For details on transferring data, refer to Operation Manual - Screen Creation Guide *Chapter 7: Transferring Data*.

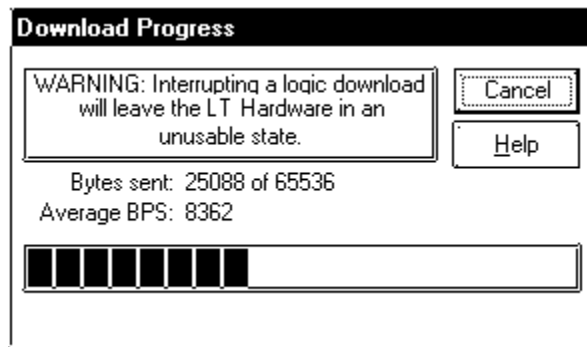
This section describes how to transfer only a logic program using the LT Editor.

#### ■ To Write to the Controller:

1. From the [Controller] menu, select [Write to Controller] and the following dialog box appears, prompting you for your OK before writing to the Controller. Before a program is written to the Controller, the LT Editor automatically runs a validity check. A program containing errors cannot be written to the Controller.



2. Click on [OK]. The [Download Progress] dialog box appears and displays the status of the download of data to the LT.



**Note:**

- The Flex Network driver software etc. will be downloaded (if needed) when you write your LTE file to the controller. If no changes in the driver have occurred since the last download, the download of the driver is skipped.
- The size of the downloaded file can be reduced by removing descriptions before transferring.

**Reference** [2.1 Configuring the LT Controller](#)

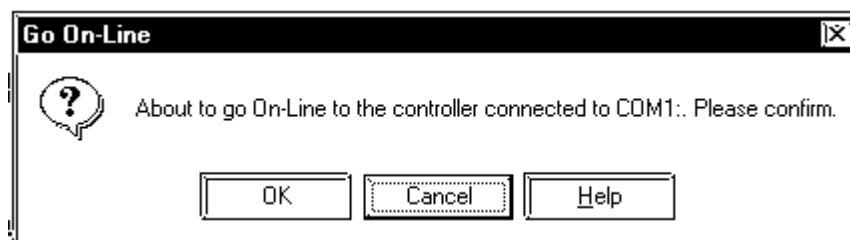


**The LT resets itself after the logic program write is completed.**

### 2.1.2 Going to Monitoring Mode

#### ■ To Go to Monitoring Mode:

1. From the [Controller] menu, select [Monitoring Mode]. A dialog box then appears asking if you wish to go online.



2. Click on [OK]. You are now online and can operate the program in the LT through the Monitoring Mode.

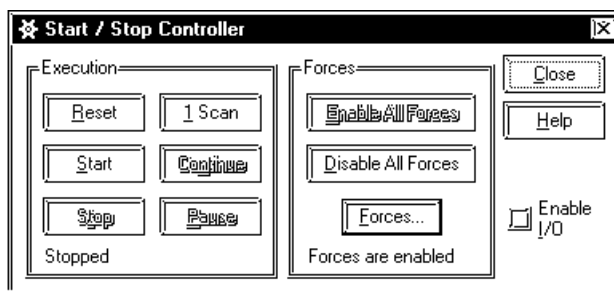
## 2.2 Starting and Stopping the Controller

When the Controller is operating in Monitoring Mode, the start/stop of the Controller can be controlled via the LT Editor. From now we will use this feature to monitor the controller's mode.

As mentioned previously, you must be online to the Controller before you can use the start/stop, or online editing functions.

### ■ To Start/Stop the Controller:

1. From the [Controller] menu, select [Start/Stop]. If you are in Programming Mode, however, this option is unavailable. The [Start/Stop Controller] window is displayed.



The functionality of the [Start/Stop Controller] window is explained below.

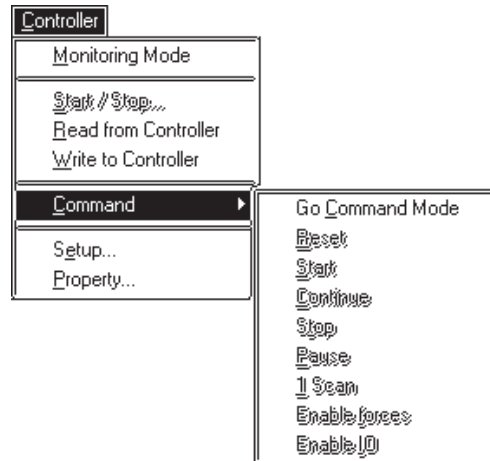
Option	Description
Start	The [Start] button starts the Controller. Once it starts, it scans from the beginning of the program and executes all logic sequentially. The first scan executes any initialization logic.
Stop	The [Stop] button stops the Controller.
Reset	The Reset button causes the Controller to reload the ".L T E" file, initialize any I/O and then stop.
1 Scan	Press this button to perform a single scan of logic. This function is useful for troubleshooting or debugging an application.
Pause	Pause button stops the Controller from scanning logic but leaves the I/O enabled.
Continue	The Continue option is available after the Pause button has been pressed. It allows the Controller to continue executing logic (or a single scan) with the current data values.
Enable All Forces	Enables the forced variables.
Disable All Forces	Disables the forced variables.
Forces	Lists all forced variables in the ladder logic program.
Enable I/O	This function enables the inputs/outputs to the LT main unit and external I/O of the I/O unit. In normal operations, the input/output of the external I/O is disabled when the LT is set to the RUN mode after performing a Logic Program download. For safety reasons, this function prevents the possibility of accidental startups of machines caused by errors in operation and logic programs.















**Note:** When the setting is changed from Start/Stop, the system internally checks the status for the "Enable IO" setting. Therefore, "Enable IO" setting changes made during the "Start" mode will not be reflected. Be sure to change the setting to "Stop" before changing the "Enable IO" setting, and then return to the "Start" mode.

## Chapter 2 – Running the Ladder Logic Program

You can also select these items from the [Controller] menu's [Command].



	Go Command Mode
	Go Online
	Write to controller
	Read from controller
	Reset
	Start
	Continue
	Stop
	Pause
	1 Scan
	Enable Forces
	Enable IO



**Note:**

If you click on [Reset], all LT Editor variables will be reset except retentive variables. Use the MOV instruction etc. if any values need special initialization.



## 2.3 Troubleshooting Using System Variables

System variables can be used to help troubleshoot for an application if it does not perform as expected.

The system variables are the most useful for detecting problems with either the Controller or the I/O are #Fault, #IOFault, #IOStatus and #ScanCount.

Option	Description
#FaultCode	#FaultCode identifies the most recent fault condition. It is reset to 0 when the first scan operates after the Controller started.
#Faultrung	#Faultrung detects the rung number which has a fault.
#IOFault	#IOFault is a discrete variable that is turned ON when a fault is detected in your I/O system.
#IOStatus	#IOStatus is an array which displays I/O specific errors. These errors are indexed with a numeric code. This code differs from driver to driver. (Reference: For a detailed explanation of the error see the driver's Help system.) An error is displayed in #IOStatus only if #IOFault has been turned ON.
#ScanCount	#ScanCount indicates the number of scans the Controller has executed since it was last started. When monitored, this variable should constantly be increasing. If it is not, the Controller is not running.

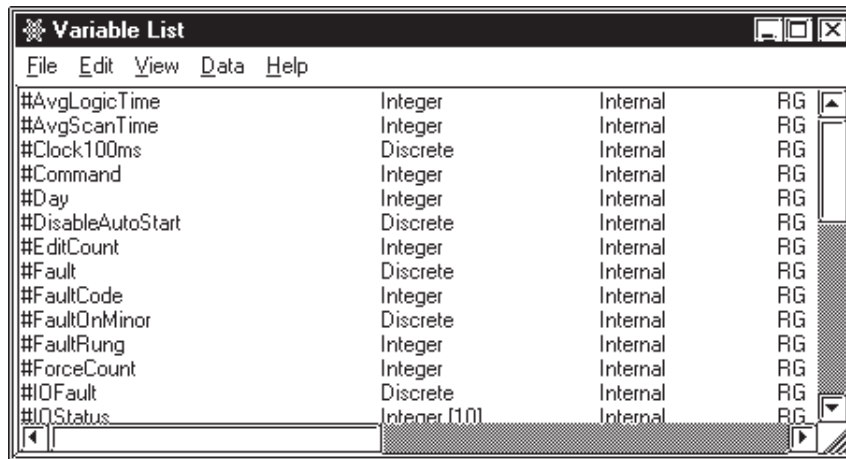
**Reference** *For details on system variables, please refer to Chapter 8 System Variables.*

## 2.4 Viewing System Variables

You can view the system variables to show information about I/O status, scan time and controller status. **Reference** *Chapter 8 System Variables*

### ■ To View System Variables:

1. From the [**Data**] menu, select [**Variable List**] and the [**Variable List**] window appears. All LT Editor system variables (variables which begin with #) should be displayed. If they are not, select [**System**] from the [**View**] menu.



2. From the [**Data**] menu, select [**Data Watch List**]. The [**Data Watch List**] window appears.
3. Click and drag the system variables you wish to monitor from the [**Variable List**] window to the [**Data Watch List**] window.

These monitored variables display the appropriate errors if they occur while the logic is being scanned.

In the following example, I/O error 821 has occurred with driver one. The #IOFault is turned ON.



## 2.5 Reading from the Controller

To edit and save a logic program located in the LT unit, read out the program from the Controller.

To read a logic program from the LT, you can either;

- receive the screen data and logic program via the "Transfer" window of the LT Editor.
- receive a ladder logic program only in the LT Editor's [Transfer] window.
- receive only a logic program via the Logic Program Editor.

This section describes how to receive a logic program exclusively using the LT Editor.

**Reference** For details on how to receive a logic program via the "Transfer" window in the LT Editor, refer to *Operation Manual - Screen Creation Guide Chapter 7 Data Transfer*.

### ■ To Read from the Controller:

1. If the Controller is online, from the [Controller] menu, select [Programming Mode].



**The Controller must be stopped before doing a "read from controller" if the program contains values that are not initialized.**

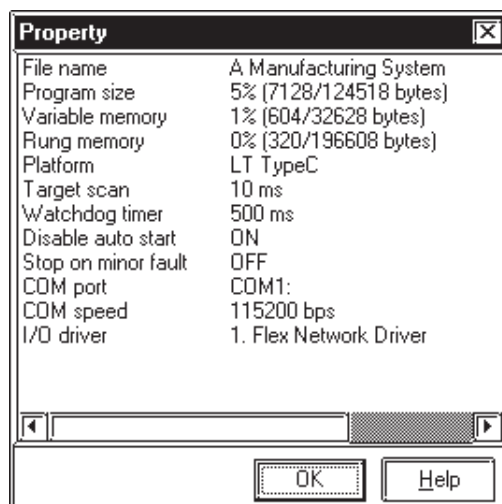
2. From the [Controller] menu, select [Read from Controller]. A copy of the program written to the Controller will be opened by the LT Editor.

You can now make changes to the program and /or save it as a "\*.lte".

## 2.6 Property

Select the [Controller] menu's [Property]. The LT program's property information list box will appear.

The [Property] box is shown below.



## 3 On-Line Editing

The LT Editor allows you to make On-Line changes to a program running in the Controller and have these changes take effect immediately. For the demonstrations and examples in this chapter use the 'Soda.lte' file, located in 'C:\Program Files\Pro-face\LT\SAMPLE'. All examples used here assume that the ladder colors and preferences use the system default.

### 3.1 Before Editing

#### ■ To execute the example program:

1. Open 'Soda.lte' file. It is included as an LT Editor sample program and is located in 'C:\Program Files\Pro-face\LT\SAMPLE'.
2. Write this program to the Controller.
3. Go On-Line to the Controller.
4. Start the Controller.

**▼Reference▲** For Controller operation, refer to “*Chapter 2 Running the Ladder Logic Program*”.

#### ■ LT Online Program Change Features

On-Line editing features are restricted for the LT platform, however, the following changes can be made to a program while it is running On-Line in the Controller.

- Turning ON/OFF discrete variables
- Integer value changes

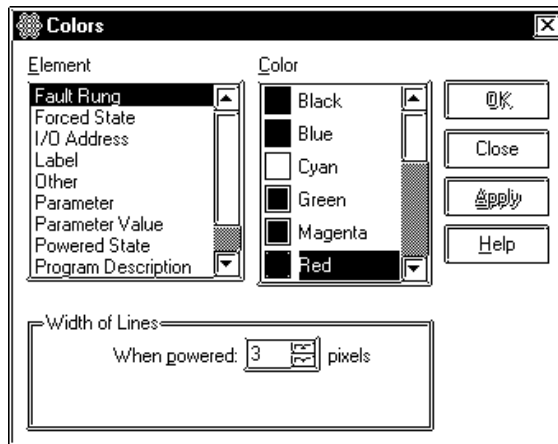
## 3.2 Using Colors for On-Line Editing

The Logic Program Editor uses default colors to indicate specific aspects and changes to a ladder logic program while running On-Line. The default colors are:

Color	Meaning
Green	Indicates circuit is charged
Red	Indicates an error has occurred.
Purple	Indicates online edit feature is being used

■ **To Change the Color Defaults in the Logic Program Editor:**

1. From the [View] menu, select [Colors] and the [Colors] dialog box appears.



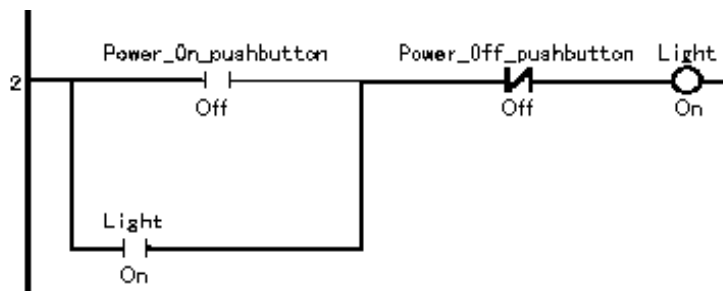
2. Select the [Element] and then the [Color] you want associated with that element, then click on [Apply].

## 3.3 Turning a Discrete ON and OFF

Discrete variables can be manually turned ON or OFF while the logic program is running. A discrete that has been turned ON is not the same as a discrete that has been forced ON, since its state can be affected by the program as it is scanned.

### ■ To Turn a Discrete ON or OFF:

1. Right-click on the variable 'Light' assigned to the output coil on rung 2.
2. Select [**Turn ON**] from the short cut menu. The 'Light' variable turns ON and the power flow indicates that power is flowing through the rung.



3. Right-click on the variable 'Light' assigned to the output coil on rung 2.
4. Select [**Turn OFF**] from the short cut menu. The 'Light' variable now turns OFF and the power flow disappears, indicating that power no longer flows through the rung.



#### Note:

Power flow is not displayed in your logic if the [Power Flow] check box is not selected in the [Monitoring] section of the [Preferences] dialog box.



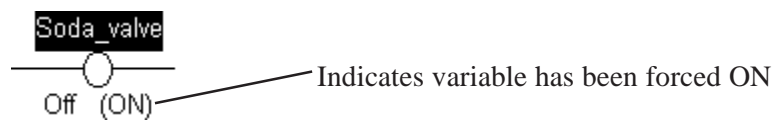
**Reference** *Chapter 1 Preference Area Settings (Prior to Creating a Logic Program)*

## 3.4 Forcing Discrete ON and OFF

A discrete can be forced ON or OFF while you are online in the Controller. The difference between turning a discrete ON or OFF and forcing it ON or OFF is that if you force it, the variable does not change its state until the force is manually changed. The program logic and I/O cannot change its state. The discrete ON and OFF operation described in section 3.3 depends on the calculation result of the program; however, the force discrete ON and OFF operation does not depend on the calculation result.

### ■ To Force a Discrete ON or OFF:

1. Right-click on the variable 'Soda\_valve' on the output coil on rung 9.
2. Select [Force ON] from the short cut menu.
3. Click on [OK] in the [Force] dialog box.



The variable turns ON and cannot be turned OFF by the ladder logic program.



#### Note:

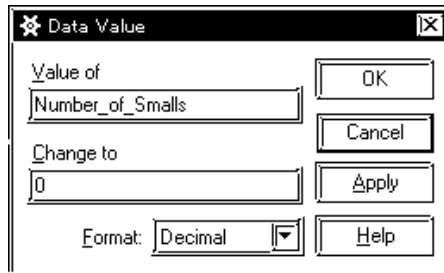
If you find that forced variables have no effect in your ladder logic program, they have probably been disabled in the LT Editor. To enable forces click on the [Enable All Forces] button in the [Start/Stop Controller] dialog box, or use the [Controller] menu and the toolbar.

## 3.5 Changing Variable Values

While you are online to the Controller you can set the value of any LT Editor variable included in your ladder logic program.

### ■ Changing a Variable Value:

1. From the [**Data**] menu, select [**Value**]. The [**Data Value**] dialog box appears.
2. Click on the variable 'Number\_of\_Smalls' in the ladder logic. The [**Data Value**] dialog box appears as follows:



3. Select the '0' in the [**Change to**] field, then type '5'.
4. Click on [**Apply**].

The value of 'Number\_of\_Smalls' is now 5. You can change other values or close the [**Data Value**] dialog box by clicking on [**Close**].



#### Note:

- You can enter data values in Decimal, Hexadecimal, Octal or Binary number format. Simply select one from the [Format] list.
- Use the [Variable List] or [Data Watch List] in conjunction with the [Data Value] dialog box to quickly find and set LT Editor variables.



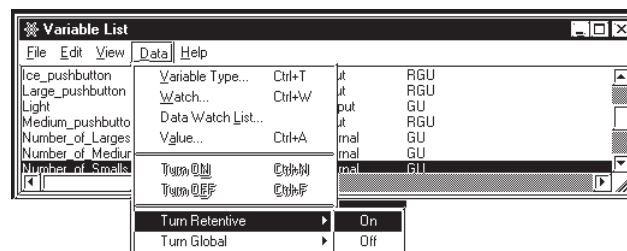
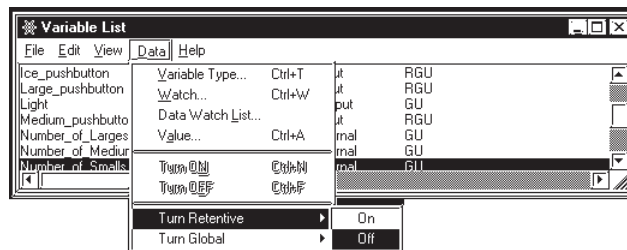
## 3.6 Changing Variable Attributes

You can use the [Data] menu to change the variable attributes (Retentive/Global.)

This menu is enabled only when you are in the programming mode.

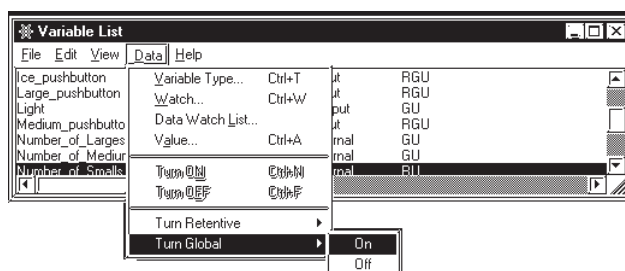
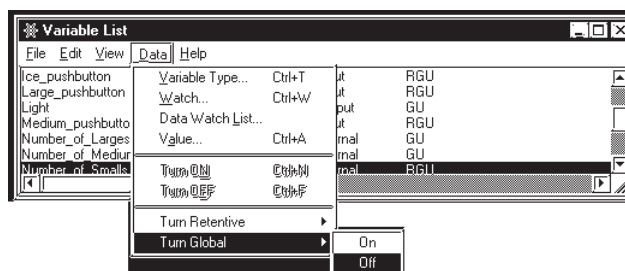
### ■ Changing a Variable Attribute (Retentive)

Select the [Data] menu's [Variable List]. The [Variable List] window will appear. Select the variable you wish to change its attribute, and change the attribute using this window as shown below. However, the system variable's "retentive" cannot be changed.



### ■ Changing a Variable Attribute (Global)

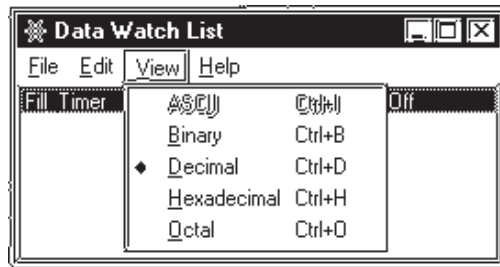
Select the [Data] menu's [Variable List]. The [Variable List] window will appear. Select the variable you wish to change its attribute, and change the attribute using this window as shown below.



## 3.7 Data Watch List

- **To change the display mode of all selected variables at the same time**

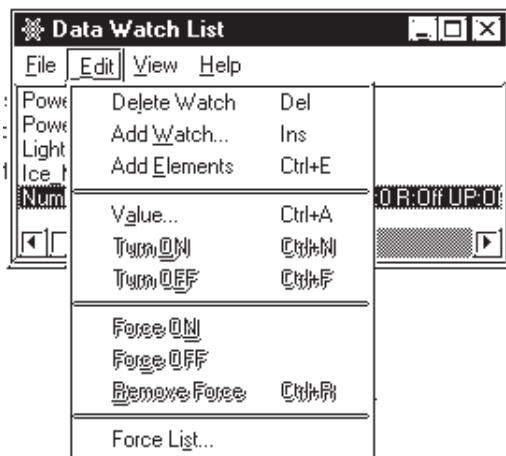
Select the [Data] menu's [Data Watch List], and select a display mode in the [View] list box. This allows you to change all of the selected variables' display mode to the designated display mode at the same time.



- **To Display Array Elements**

When creating an array via [Data Watch List], you can display array counter/timer's values by element.

1. Select the [Data] menu's [Variable List] and [Data Watch List].
2. Select the [Data Watch List] menu's [Edit], and select [Add Elements] from the [Edit] box.



# 4 Errors and Warnings

Error or warning displays may appear in the [Validity] dialog box when a validity check is done on a program. These errors and warnings may be related to a problem with the program's logic, variables or I/O. The errors are indexed numerically, with each numeral being part of a specific range. Each range specifies a general area for you to focus on when determining why the error or warning has occurred.

## 200-299: Logic errors and warnings

**Reference** *For information on instruction for the logic program, select it in the main window, and then from the [Help] menu select [Context], or press the [F1] key.*

### ◆ Error 200 – Parameter should be a Discrete

The instruction requires a Discrete operand. This can be:

- A Discrete variable,
- An element of a Discrete array, or
- A Discrete element of an Integer variable.

### ◆ Error 201 – Parameter should be a Counter

The instruction requires a Counter variable.

### ◆ Error 202 – Parameter should be a Timer

The instruction requires a Timer variable.

### ◆ Error 203 – Parameter should be an Integer or Real

The instruction requires an Integer or Real, either as a variable or a constant.

### ◆ Error 204 – Parameter should be a non-constant Integer or Real

The instruction requires an Integer or Real variable. It cannot be a constant.

### ◆ Error 205 - Parameter should be an Integer

The instruction requires an Integer as a variable or a constant.

### ◆ Error 206 – Parameter should be an Integer but not an array

The instruction requires an Integer, either as a variable or a constant. It cannot be an array.

### ◆ Error 207 – Parameter should be a non-constant Integer

The instruction requires an Integer variable. It cannot be a constant.

### ◆ Error 208 – Parameter should be a label

The instruction requires a label name, and a label with that name must exist.

### ◆ Error 209 – Parameter should be a subroutine

The instruction requires a subroutine name.

### ◆ Error 210 – Label is out of scope

The specified label exists, but cannot be reached from here.

## Chapter 4 – Errors and Warnings

- ◆ **Error 211- Subroutine cannot call itself**

The Jump Subroutine instruction is attempting to call the subroutine that contains it. This is not allowed.
- ◆ **Error 212 – X should be the same type as Y**

The two parameters should have the same type (Integer, Real, etc.).
- ◆ **Error 213 – X should be the same size as Y**

The two parameters must be the same size. That is, both must be either:

  - Arrays with the same number of elements, or
  - Non-arrays.
- ◆ **Error 214 – X should be the same size as Y or be an Integer.**

The two parameters must be the same size or the second can be an Integer that is treated as if it is the larger size.
- ◆ **Error 215 – X should be an Integer, a Real or a Discrete array**

The instruction requires an Integer, Real or Discrete, either as a simple variable or a complete array.
- ◆ **Error 216 – X should be a non-constant Integer, Real or Discrete array.**

The instruction requires an Integer, Real or Discrete, either as a simple variable or a complete array. It cannot be a constant.
- ◆ **Warning 217 – Both parameters are constants**

The instruction is comparing two constants.
- ◆ **Warning 218 – Input parameter used on output instruction**

The variable is marked as an input (refer to [Variable Type] window), however, it is used in an output instruction. Double-check its I/O assignment.
- ◆ **Warning 219 – Preset value is zero**

The preset value of the counter is set to zero.
- ◆ **Warning 220 – Preset time is zero**

The preset time of the timer is set to zero.
- ◆ **Warning 224 – Parameter should not be retentive**

The variables assigned to the instruction parameter cannot be “Hold” type.
- ◆ **Warning 225 – X should be an Integer Array**

The instruction requires Integer as a complete array.
- ◆ **Error 250 – Duplicate labels are not allowed**

The same label is defined more than once. This is not allowed, even in different sections of the program.
- ◆ **Warning 251 – Empty subroutines have no effect**

The subroutine contains no rungs. If you do not alter the empty subroutine it will have no effect on your program.

◆ **Warning 252 – Empty rungs have no effect**

The rung contains no instructions. If you do not alter the empty rung it will have no effect on your program.

◆ **Warning 253 – Empty branches have no effect**

The branch contains no instructions. If you do not alter the empty branch it will have no effect on your program.

◆ **Error 254 – Control instruction should be last on rung.**

The instruction cannot have any others to the right of it.

◆ **Warning 255 – X is used by more than one timer instruction**

The timer variable is used by more than one timer instruction. The results are indefinite.

▼ **Reference** ▲ You can use the [References] window to find the other instruction(s).

◆ **Error 256 – X is used by more than one counter instruction**

The Counter variable is used by more than one counter instruction. The results are indefinite.

▼ **Reference** ▲ You can use the [References] window to find the other instruction(s).

◆ **Error 257 – Last instruction on rung should be an output**

The instruction is not an output instruction (i.e., it does not change the values of its parameters).

◆ **Error 258 – Multiple outputs are not allowed**

An output instruction cannot have other instructions to the right of it.

◆ **Error 259 – Last instruction on branch should be an output**

An output instruction cannot have other instructions to the right of it.

◆ **Error 260 – Maximum level of nesting exceeded**

The rung has too many levels of branches (the maximum number of levels is 25). Try dividing the rung into several smaller ones.

◆ **Error 262 – Program is too large (by xx %), see Controller | Setup | Memory**

The program size is larger than the LT Flash Memory.

◆ **Warning 263**

The variable is used by more than one coil. When the ladder logic program is executed, the result of the last instruction to which the variable is assigned will be effective.

◆ **Error 269**

The rung memory usage has been exceeded by xx%.

◆ **Error 270**

The maximum label number has been exceeded. The maximum number is 2048.

◆ **Error 271**

The maximum variable number has been exceeded. The maximum number is 8192.

## Chapter 4 – Errors and Warnings

### ◆ Error 272

The maximum constant number has been exceeded.

### ◆ Error 273

The maximum numbers of NT instructions and PT instructions have been exceeded. The maximum number is 2048.

### ◆ Error 274

The maximum numbers of Advanced instructions was exceeded. The maximum number is 100.

## 300-399: Variable errors and warnings

### ◆ Warning 300 – Variable has input or output type but no I/O address assigned

The variable is marked as an input or output (refer to the [Variable Type] window), however, it is not mapped to any I/O.

### ◆ Error 301 – Type not assigned

The variable has not been assigned a variable type. To assign a variable type use the [Variable Type] window.

### ◆ Error 302 – Label not found

The Jump Subroutine instruction refers to a label that does not exist.

### ◆ Error 303 – Variable referenced should be a Timer or Counter

You have specified an element of a Timer or Counter variable, however, the variable is actually of a different type. Refer to the [Variable Type] window.

### ◆ Error 304 – Variable(s) referenced should be Integer type

You have used a variable to specify an array element or modifier. This variable must be an Integer. Refer to the [Variable Type] window.

### ◆ Error 305 – Array reference to non-array variable

You have specified an element of an array, however, the variable is not designated as an array. Refer to the [Variable Type] window.

### ◆ Error 306 – Array reference is beyond size of array

You have specified an element of an array using a constant that is equal to or larger than the array's size. (Note that the valid elements are numbered 0 to size-1). You can change the size in the [Variable Type] window.

### ◆ Error 308 – Modifier reference is out of range

You have specified a bit, byte or word element that is out of range.

### ◆ Error 309 – Reference is invalid for the variable

You have specified a timer reference for a counter variable, or vice versa.

### ◆ Warning 310 – ...Already exists and will be replaced

A variable by that name already exists. The new one will replace the original one if you click on [OK] in the [Variable Import Status] window.

### ◆ **Error 311 – The clipboard buffer is not a recognized format**

The current contents of the clipboard are not suitable for pasting into the [Variable List] window.

### ◆ **Error 312 – Too many warnings**

The [Variable Import Status] window only shows a certain number of warnings. If you see this message, there may be more warnings that does not show.

### ◆ **Warning 313 – Missing ]**

An array type requires the size enclosed in square (“[ ]”)brackets. For example, Integer [10].

### ◆ **Warning 314 – Array size is invalid ...Assuming a size of 1**

This variable apparently is intended to be an array, however, the size is not recognizable. The size should be an integer within square brackets. For example, Integer [10].

### ◆ **Warning 315 – Unknown type ...will be Not Assigned**

The text is not recognized as a LT Editor variable type. Possible causes are:

1. It is spelled incorrectly
2. It has leading and / or trailing blanks.

### ◆ **Warning 316 – Unsupported array type ... Ignoring array settings**

That variable cannot be an array.

### ◆ **Error 317 – Invalid variable name...**

You have entered an invalid variable name.

### ◆ **Error 318 – Too many errors**

The [Variable Import Status] window only shows a certain number of errors. If you see this message, there may be more that it does not show.

### ◆ **Error 320 – Too many variables**

You have attempted to assign too many variables.

### ◆ **Error 321 – Too many variables**

You have attempted to assign too many variables. Reduce the number of variables.

## **400-499: Logic Program LT Editor I/O errors and warnings**

### ◆ **Error 400 – Variable Name has already been mapped**

The variable is mapped to more than one I/O point. Refer to the [Configure I/O] window.

## **500-549: Generic I/O driver errors**

### ◆ **Error 500**

The LTE file may be corrupted or was damaged during download.

### ◆ **Error 501 – Internal variable mapped to I/O terminal**

The variable is marked as “internal”, however, it is mapped to an I/O terminal. Refer to the [Variable Type] window.

## Chapter 4 – Errors and Warnings

### ◆ Error 502 – Input variable mapped to output terminal

The variable is marked as an input, however, it is mapped to an output terminal. Refer to the [Variable Type] window.

### ◆ Error 503 – Output variable mapped to input terminal

The variable is marked as an output, however, it is mapped to an input terminal. Refer to the [Variable Type] window.

### ◆ Error 504 – Discrete variable mapped to analog terminal

A Discrete variable cannot be mapped to an analog terminal.

### ◆ Error 505 – Integer variable mapped to discrete terminal

An Integer variable cannot be mapped to a discrete terminal.

### ◆ Error 506 – Variable type not supported by I/O driver

The I/O driver requires a different type of variable to be mapped to this terminal.

### ◆ Error 507

Variables have not been allocated to terminals.

### ◆ Error 508

LT type has been selected that is not supported by the current set of drivers.

## 600-799: PID Instruction Error

### ◆ Error 600

Set a control block variable to the Integer array with at least seven elements.

### ◆ Error 601

Set a special variable “Variable.SP” to the Integer.

## 800-899: Specific I/O driver errors

**▼Reference▲** *For information about any errors pertaining to your I/O driver, refer to your I/O driver user guide.*

## 900-1000: Specific I/O driver warnings

**▼Reference▲** *For information about any warnings pertaining to your I/O driver, refer to your I/O driver’s User Guide.*



# 5 Glossary of Terms

## ■ **Array**

A Discrete, Integer or Real variable can be designated as an array. This means that multiple elements of that type are allocated under a single name.

## ■ **Bit**

The basic storage element, its value may be either 1 or 0.

## ■ **Bookmark**

An invisible marker that can be placed anywhere in your logic, allowing you to instantly return to that portion of your program.

## ■ **Branch**

A parallel path of execution on a rung.

## ■ **Byte**

A storage element containing 8 bits of information. A byte may be assigned values from 0 to 255. A Logic Program Editor integer is composed of 4 bytes.

## ■ **Clipboard**

A temporary storage place maintained by Windows for copying and pasting data. This can be done between applications or within a single application.

## ■ **Data Watch List Window**

Shows data values as they change. You can adjust the update rate in the [**Preferences**] dialog box.

## ■ **Descriptions**

A description can be any amount of text, up to 32767 single-byte characters, that describes some part of your program. A summary of descriptions may be viewed with the [**Description List**] window.

## ■ **Discrete point**

A point that can have one of two states: OFF or ON.

## ■ **Drag and Drop**

To press and hold down the left mouse button, move the mouse, then release. The mouse pointer indicates whether this is a valid place to let go.

## Chapter 5 – Glossary of Terms

### ■ **Element**

An element is a name for some part of a variable, rather than the whole thing. This part can be:

- An element of a Timer or Counter variable,
- An element of an array, or
- Part of an Integer

### ■ **Error (Fault Conditions)**

There are three types: **Major**, **Minor**, and **I/O**.

A Major Fault is serious. When this occurs, the Controller stops executing logic immediately. The editor shows the state as “**MAJOR FAULT**”. To clear the condition, the Controller must be reset using the [**Start/Stop**] window.

A Minor Fault is one that can be safely ignored.

An I/O Fault is a failure to read or write I/O in.

### ■ **Focus**

A black rectangle that highlights a selection in the ladder logic program.

### ■ **Forces**

Discrete points can be forced either ON or OFF. This overrides any actions the logic may take. For example, if a variable is forced OFF, but the logic is trying to turn it on, it stays off. A list of the forces in your program can be viewed with the [**Force List**] window.

### ■ **LT Controller**

The LT Controller executes ladder logic and controls I/O. The Controller is invisible and performs the LT unit’s extended tasks. The Logic Program Editor monitors the controller in Monitoring Mode.

### ■ **Hexadecimal**

A base-16 representation of an integer value. These can be entered with 16# in front. For example, 16#FF is 255.

### ■ **IEC 61131-3**

A standard developed by the International Electrotechnical Commission defining the printed and displayed representation of five control languages including: Instruction List (IL), Ladder Logic Diagrams (LD), Function Block Diagrams (FBD), Structured Text (ST), and Sequential Function Charts (SFC).

The smallest component in a rung which instructs the LT Editor Controller to perform a specific function (i.e., Discrete, Bit operand, Data control, Operand, Timer/Counter, and Program control instructions). Instructions in LT Editor are based on the IEC 61131-3 specification.

### ■ **Instruction**

The smallest component in a rung which instructs the LT Controller to perform a specific function (i.e., Discrete, Bit operand, Data control, Operand, Timer/Counter, and Program control instructions). Instructions in Logic Program Editor are based on the IEC 61131-3 specification.

### ■ **Integer**

A storage element containing 32 bits of information. An integer may be assigned values ranging from  $-2147483648$  to  $2147483647$  ( $16\#00000000$  to  $16\#FFFFFFF$  in hexadecimal). Integers cannot contain decimal points.

### ■ **Internal Variable**

A variable that is not mapped to an I/O point.

### ■ **I/O**

Input/Output. The LT Controller connects to physical (real-world) devices through I/O hardware supplied by third parties.

### ■ **I/O Address**

An address assigned to a variable when it is mapped to an I/O device. The format of an I/O address depends on the driver it is mapped to.

### ■ **Label Name**

A name containing up to 32 characters that identify or label a position within the ladder logic. It cannot start with a digit.

### ■ **Ladder Logic**

The collection of rungs that make up your application. So called because it looks vaguely like a ladder.

### ■ **Off-Line**

When Off-Line, the LT Editor works with the disk file '.lte' containing a ladder logic program. This program is developed Off-Line and then run On-Line with the Controller.

### ■ **On-Line**

The LT Editor monitors a program which is running 'live' with the LT Controller. For example; Power\_off\_pushbutton, ResetButton, ALARM2 etc.

### ■ **Parameter**

An input to or output from an instruction. Parameters are entered into the Instruction Parameter Box.

### ■ **Power Flow**

The path power is taking through the ladder logic program.

## Chapter 5 – Glossary of Terms

### ■ Real

Any number containing a decimal point or being represented in scientific notation. The range for a real in Logic Program Editor is  $+2.25e^{-308}$  to  $+1.79e^{-308}$ . It can have up to 15 significant digits.

### ■ State Flow

Highlights individual instructions based on their parameters. Each contact is highlighted if it is able to pass power (as opposed to whether it actually gets power), based on the state of its parameter.

### ■ Subroutine

A group of rungs in a separate, named area.

Subroutines are placed between the END and PEND (Program End) markers, and cannot be placed within other subroutines. When you click on [**Subroutine**] from the [**Insert**] menu, both a “Subroutine Start” and a “Subroutine End” markers are created. You can then insert logic between the two.

Subroutine are called with a “Jump Subroutine (JSR)” instruction. The advantage is that they can be called from many places, and the code only needs to be written once. A subroutine name is required.

### ■ Subroutine Name

A Subroutine Name consists of up to 32 letters, digits, and / or underscores. It can only start with a letter.

### ■ System Variables

System Variables are special, predefined variables that provide information about the controller’s status or affect its operation. They perform like ordinary variables, except that they are created automatically and cannot be deleted.

### ■ Variable

Storage locations for data values are called variables. Easy-to-understand names are recommended to use, rather than using numbered addresses. A variable name is up to 20 letters, digits, and / or underscores. It cannot start with a digit. Some valid examples are: Power\_Off\_pushbutton, ResetButton and ALARM2, etc. LT Editor creates an appropriate type of variable automatically as soon as a new variable name is entered either in [**Parameter Box**] or the [**Configure I/O**] window.

### ■ Watchdog Timer

Detects an error if the program did not finish running up to the “END” rung within a certain length of time. To set “Watchdog Timer”, select [**Setup**] from the [**Controller**] menu, and enter time in millisecond in the [**Watchdog Timer**] box in the [**Tuning**] tab.

### ■ Word

A storage element containing 16 bits of information. A word may be assigned values ranging from 0 to 65535.



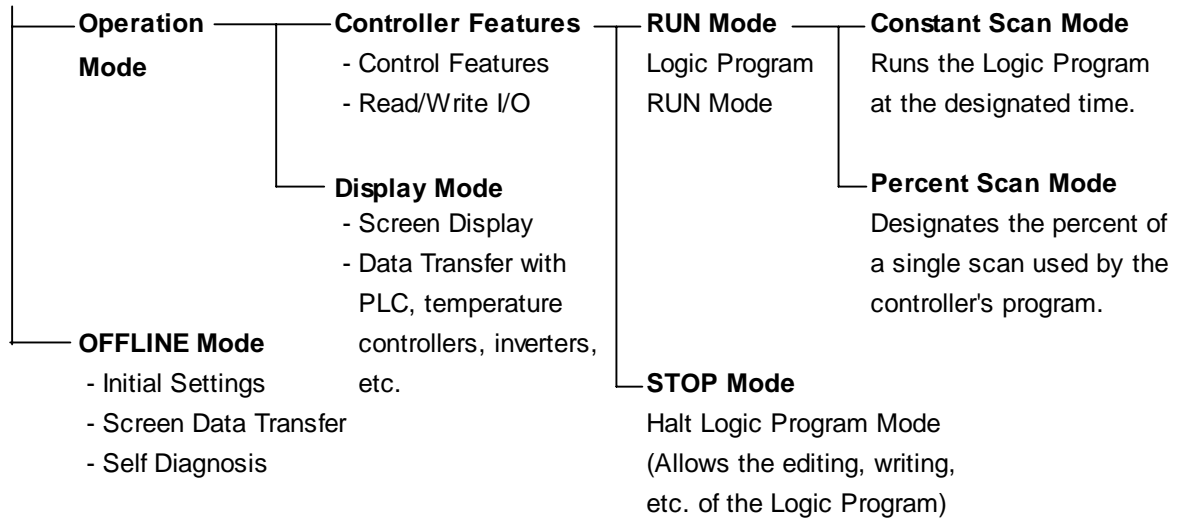
# ***Features***

# 6 Controller Features

## 6.1 Operating the LT

The LT contains both screen display and I/O control features. These features and their respective modes are described below.

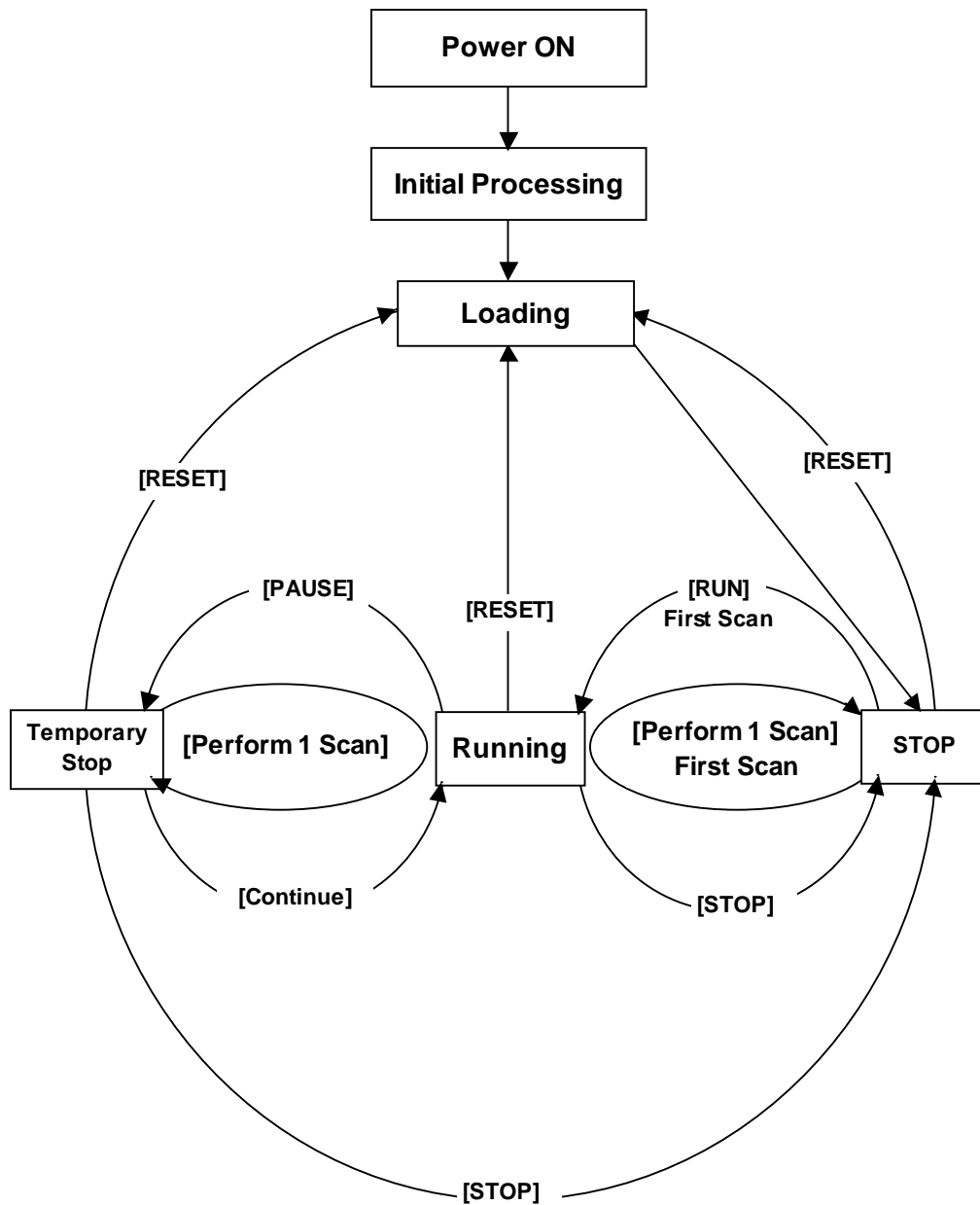
### LT Features



- ***Understanding the LT's operation modes is critical for designing a system. Please read this chapter thoroughly to understand the operation and design the system in consideration of safety issues.***
- ***When OFFLINE mode is entered, the controller will stop. Re-entering RUN mode will reset the LT.***

**6.1.1 Controller Feature Overview**

The Controller feature functions as follows. The facing page provides detailed descriptions of each step.



■ **Initial Processing**

This is the original state of the engine use to perform the Logic Program. Once initialization is finished, the Controller enters the “Loading” state.

■ **Loading**

Here, the actual reading in from memory of the Logic Program is performed. After a check is performed to determine whether the Logic Program is successfully loaded or not, error processing is performed if an error has occurred. If Loading is successful, the program enters the [STOP] state. If the [Power ON Operation Mode] is set to [START], the [RUN] instruction is automatically performed.

### ■ STOP

In this condition the Controller is waiting to receive another instruction. Once the [RESET], [Perform 1 Scan], [Continue], or [PAUSE] instructions are received, the Controller changes to that condition.

The [RESET] instruction will change the program to the [Loading] condition.

At this time, variables are initialized. Retentive variables maintain data before the power shuts down or the LT resets. However, when triggering Controller reset in Monitoring mode (\*1), or when using the #Command, the value set in Programming Mode (\*2) is used as the initial value. Non-retentive variables are cleared to zero.

The [RUN] instruction will change the program to the [Running] condition.

The [Perform 1 Scan] instruction will perform the program once.

### ■ First Scan

Executes the I/O Read, performs any Logic Program that is higher the START level, and executes the I/O write.

### ■ Running

This is the Logic Program performance engine's continuous performance mode. In this mode, it executes I/O Reads, performs Logic Programs, executes I/O writes, and updates System Variables. (#AvgLogicTime, #AvgScanTime, etc.)

The [RESET] instruction will change the program to the [Loading] condition.

The [STOP] instruction will change the program to the [STOP] condition.

The [PAUSE] instruction will change the program to the [Temporary Stop] condition.

### ■ Temporary Stop

The logic program execution engine is temporarily stopped in this state. To avoid an I/O watchdog timeout, the system executes an I/O read and I/O write. However, the logic program is not executed, so the output state does not change. When a command is received, the system switches to the appropriate state.

The [RESET] instruction will change the program to the [Loading] condition.

The [Perform 1 Scan] instruction will perform the program once.

The [STOP] instruction will change the program to the [STOP] condition.

The [Continue] instruction will change the program to the [Running] condition.

---

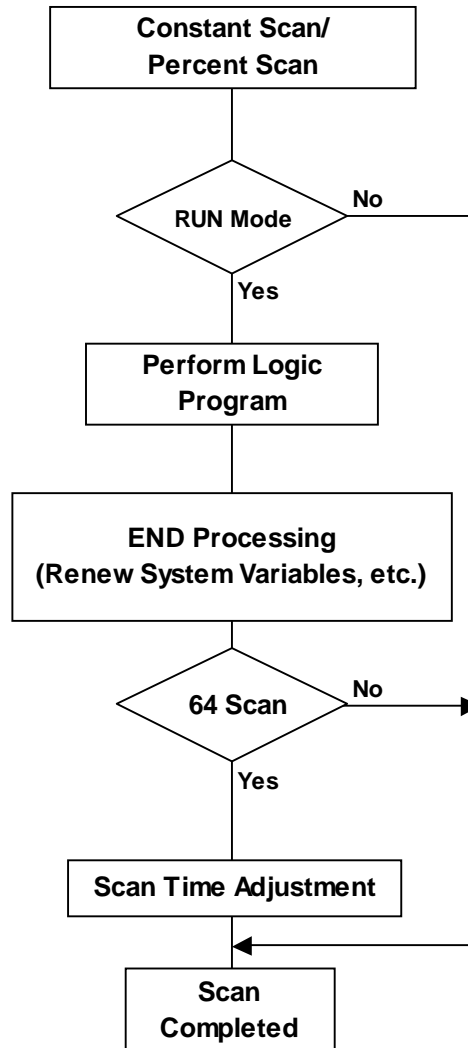
1. Mode used edit the program currently being executed by the controller.

2. Mode used to create a program.



**6.1.2 RUN Mode**

RUN Mode uses the following steps.



■ **Scan Time Adjustment**

This adjustment is performed every 64 scans. The various types of adjustments are described below for Constant Scan Time, and Percent Scan Time.

◆ **Constant Scan Time Mode**

$$LT \text{ scan time} = (\#AvgLogicTime \times 100) / 50$$

◆ **Percent Scan Time Mode**

$$LT \text{ scan time} = (\#AvgLogicTime \times 100) / \#PercentAlloc$$

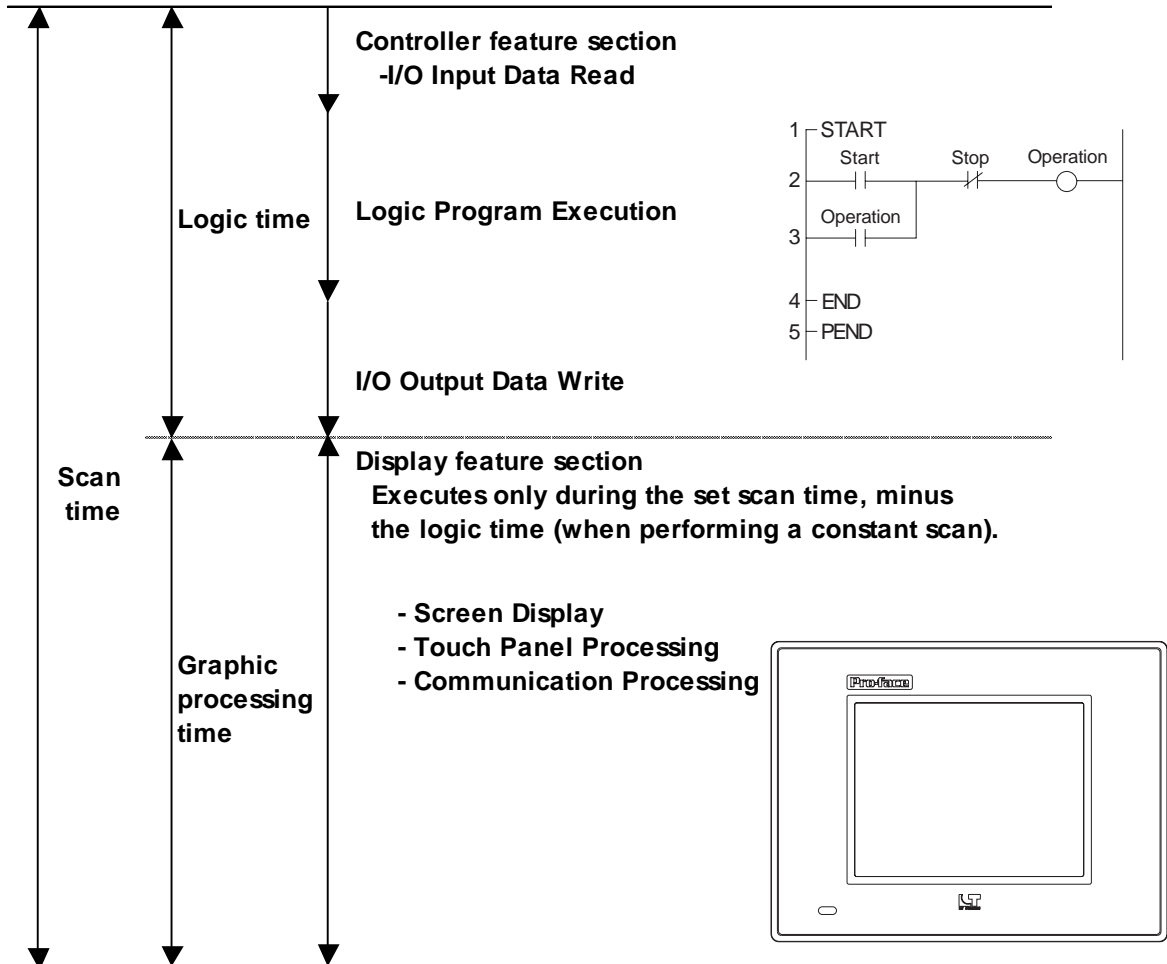
▼ **Reference** ▼ For information about #AvgLogicTime or #PercentAlloc, *Chapter 8 System Variables*



**The LT's Scan Time includes an error of approximately +0.3%.**

**6.1.3 LT Scan Overview**

LT Scan time has two modes, Constant Scan time mode and Percent Scan time mode. Their basic scan time includes two parts, logic program execution controller and display (screen/touch panel processing time, external device processing time) as follows.

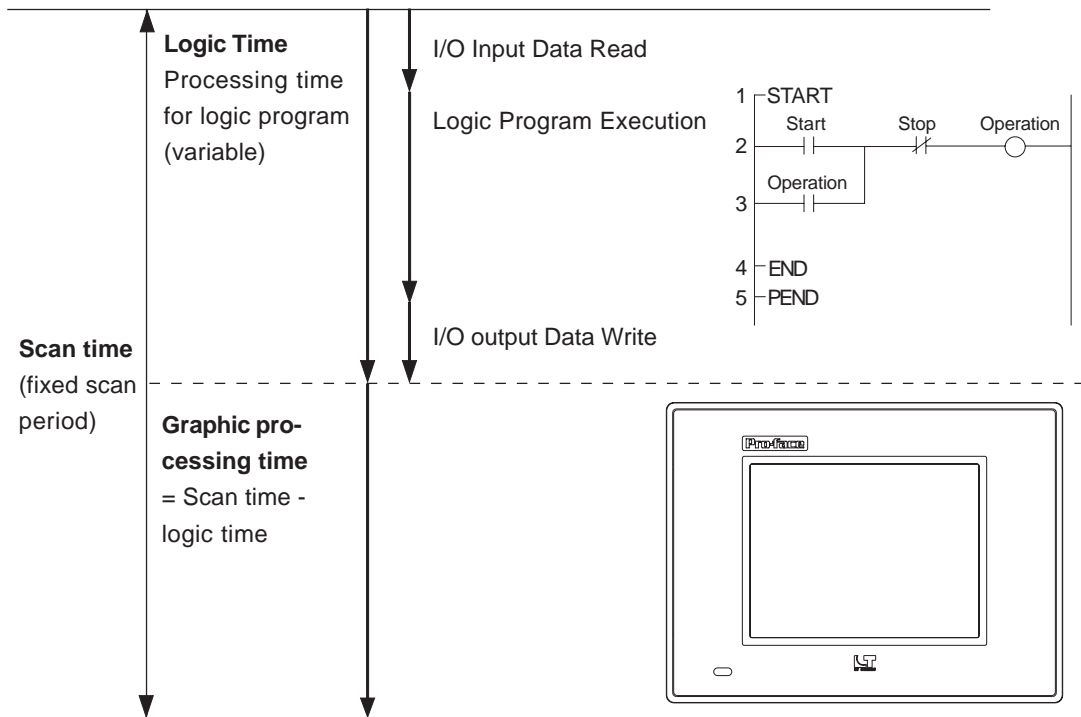


**Reference** 6.1.2 RUN Mode

### ■ Constant Scan Mode

This mode constantly executes the program during the scan time set.

When this setting is used, the screen is used mainly for data display and less for operation, with control (logic program) being the priority.



Graphic processing time = Setting time for constant scan time mode (ms) - logic time (variable)

e.g.) If constant scan time is set to 50ms and logic executing time is 20ms

$$\begin{aligned} \text{Graphic processing time} &= 50\text{ms} - 20\text{ms} \\ &= 30\text{ms} \end{aligned}$$

The longer the logic time, the shorter the graphic processing time will become. Though LT display response will be slower, the logic program will execute constantly.



**When the logic time exceeds 50% of the designated setting value for constant scan mode, the scan time is automatically adjusted so that it is twice as long as the logic time.**

**Example) When the logic time is 30 ms and the constant scan mode is 50 ms, the scan time is 60 ms.**



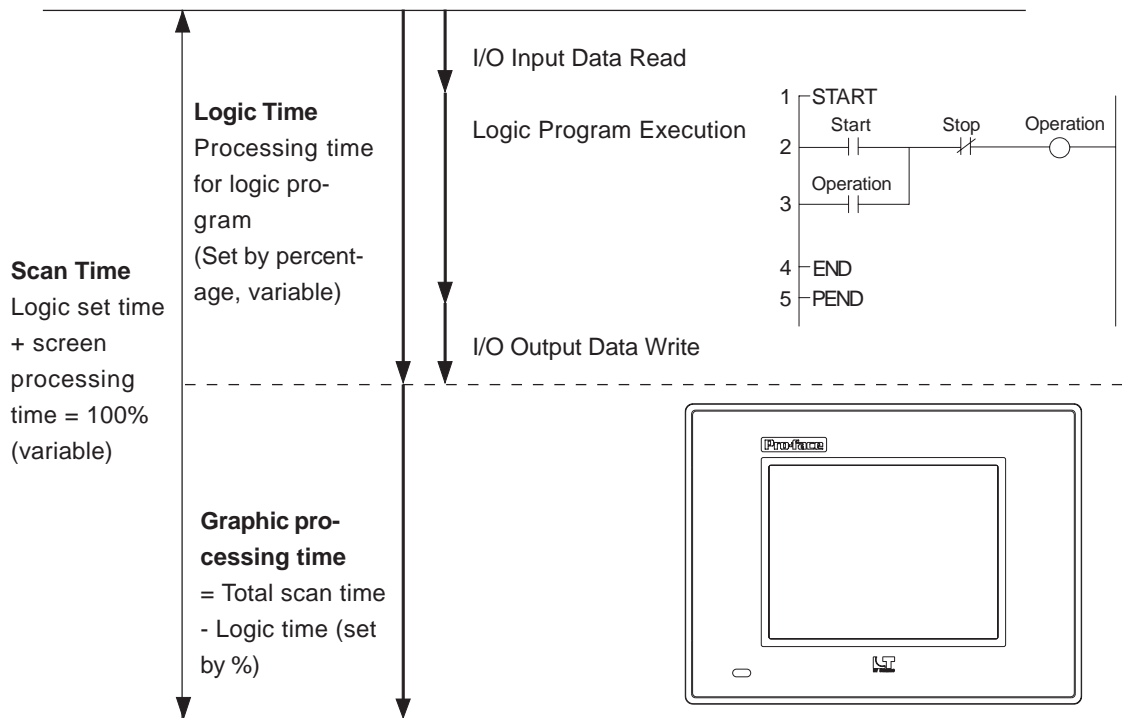
- Make sure to enter the setting value for the scan time in 10-ms increments.
- When determining the value for the setting time, use the #AvgScanTime value obtained from a test run of the LT.

**Reference** 8.2.2 #AvgScanTime

### ■ Percent Scan Mode

This mode varies the scan time according to the percentage set by the logic time

This feature sets the priority to screen operation speed and screen switching speed required during Logic Program execution.



$$\text{Scan time} = \text{Logic time} / \text{Percent scan set time} (\%)$$

e.g.) If percent scan time is set to 40% and logic executing time is 20ms

$$\begin{aligned} \text{Scan time} &= (20 \div 40) \times 100 \\ &= 50\text{ms} \end{aligned}$$

$$\begin{aligned} \text{Graphic processing time} &= 50\text{ms} - 20\text{ms} \\ &= 30\text{ms} \end{aligned}$$

When logic time increases, display processing time also increases, resulting in longer scan times.

The longer the logic time, the longer the time allocated to display processing. Therefore, the display is updated more quickly, however, the logic program processing cycle slows.



- ***There is no change in the processing time for one instruction in the logic program.***
- ***The scan setting (%) cannot be set to more than 50%.***
- ***When the percent scan setting is set to 50%, the display and logic program are processed at the same time. The display process will not be given priority.***



**Set the percent scan value so that the scan time is set in 10-ms increments.**

# *Memo*

# 7 Variables

This chapter explains the different types of variables used by the LT software.

Using hardware-independent variables enhances the reusability of your programs.

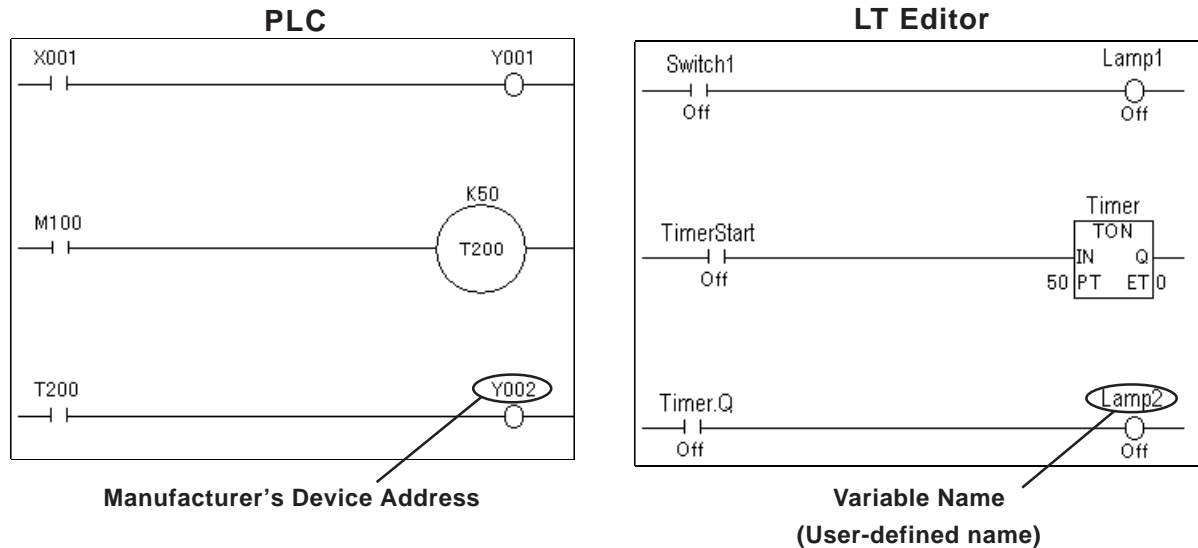
## 7.1 Variable Names

LT Editor uses variables to store I/O and counter data. Variables settings and names are user designated and are used as-is in the logic program.

In a conventional PLC, the area used to store data is called a device address. These addresses are given specific names by each PLC manufacturer.

PLC Manufacturer	External I/O	Internal Relay	Timer	Data Register
Mitsubishi	X001	M100	T200	D00001
Omron	01	1001	TIM000	DM0000
Digital Electronics Corporation	Switch1	Timerstart	Timer	Operating Time, etc.

With the LT Editor, you can assign names to these device addresses and use them as **variables** in the logic program.



**Note:**

In LT Editor (Ver. 1.03 or later), the preprogrammed variables listed below are registered in the following files in the Program Files\Pro-face\LT\SAMPLE folder: Variable Sample I (TypeA): lte; Variable Sample II (TypeA): lte; and Variable Sample III (TypeA): lte.

## Chapter 7 – Variables

Variable Sample I (TypeA): lte

Device type	Variable name	Variable type	Variable attributes
Internal relay	M0 to M127	Discrete	Non-retentive/Global/Internal
Counter	C0 to C63	Counter	Retentive/Global/Internal
Timer	T0 to T63	Timer	Retentive/Global/Internal
Register	D0 to D127	Integer	Retentive/Global/Internal
Retentive relay	L0 to L127	Discrete	Retentive/Global/Internal
Input relay	X0 to L127	Discrete	Non-retentive/Global/Input
Output relay	X0 to X15	Discrete	Non-retentive/Global/Output

Variable Sample II (TypeA): lte

Device type	Variable name	Variable type	Variable attributes
Internal relay	M[0] to M[255]	Discrete	Non-retentive/Non-global/Internal
Counter	C0 to C31	Counter	Non-retentive/Global/Internal
Timer	T0 to T31	Timer	Non-retentive/Global/Internal
Register	D[0] to D[255]	Integer	Non-retentive/Non-global/Internal
Retentive relay	L[0] to L[255]	Discrete	Retentive/Non-global/Internal
Link relay	B[0] to B[127]	Discrete	Non-retentive/Global/Internal
Link register	W[0] to W[127]	Integer	Non-retentive/Global/Internal
Input relay	X[0] to L[15]	Discrete	Non-retentive/Non-global/Input
Output relay	Y[0] to Y[15]	Discrete	Non-retentive/Non-Global/Output

Variable Sample III (TypeA): lte

Device type	Variable name	Variable type	Variable attributes
Internal relay	R[0] to R[255]	Discrete	Non-retentive/Non-global/Internal
Counter	CNT0 to CNT31	Counter	Non-retentive/Global/Internal
Timer	TIM0 to TIM31	Timer	Non-retentive/Global/Internal
Register	DM[0] to DM[511]	Integer	Retentive/Global/Internal
Retentive relay	HR[0] to HR[255]	Discrete	Retentive/Non-global/Internal
Link relay	LR[0] to LR[127]	Discrete	Non-retentive/Global/Internal
Input relay	IN[0] to IN[15]	Discrete	Non-retentive/Global/Input
Output relay	OUT [0] to OUT [15]	Discrete	Non-retentive/Global/Output

Variable names can be designated by the user. When designating variable names, be aware of the following limitations.

- Maximum Variable Name length is 20 characters (20 bytes).  
Each array's element and timer and each counter's special variables ([.PT], [.PV], etc.) are also counted as a character.
- No differentiation is made between upper- and lower-case characters. If duplicates are created, only the first word registered will be enabled (valid).  
E.g.: If "TANK" was registered before "tank," "tank" will be invalid, even though entering it will not create an error.
- Except for the first character, variable names can use numbers.
- Variable names cannot contain spaces.

- The underscore ( \_ ) is the only special character that can be used. However, underscores ( \_ \_ ) cannot be used consecutively (OK: tank\_1; Not OK: tank\_ \_1).
- The “#” sign cannot be used since it is a reserved character, .
- “LS” and “LSS” are reserved names for use by the LT unit’s system in the System Data Area, the Read Area, and for Special Relays. Therefore, they cannot be used for variable names.

**Reference** *Chapter 10 LS Area Refresh*



**Note:** When creating variable names, Pro-face recommends using the underscore character to divide the variables into blocks, or groups. This will make the variable names easier to find.

E.g.: If you have several conveyor belts in your factory system (Conveyor A, Conveyor B, Conveyor C, etc.), include an identifying character in the motor and sensor variable names:

Conveyor A variables:

A\_Motor

A\_Sensor

You could also name a Discrete (bit) as B, Integer as I, floating point as F:

AB\_MotorStartingSwitch

AI\_MotorRotationNumber

AF\_MotorPowerRatio

Here, the variables used for contacts and coils are distinguished from the variables used for basic mathematical operations.

- You can also use an array to set up variable names for each of your PLC’s devices.

Example

PLC Device	LT Editor	
	Array Variable	Variable Type
External Input	X[100]	Discrete
External Output	Y[100]	Discrete
Internal Relay	M[100]	Discrete
Data Register	D[100]	Integer

**Reference** *For information about Variable Settings, refer to 1.2 Creating Variables.*

*For information about reserved System Variables, refer to Chapter 8 System Variables.*



## 7.2 Variable Types

The Logic Program Editor uses three types of variables — Discrete (bit), Integer, and Real. Within these types, Timers and Counters are also used. Arrays can be defined and used with each type of variable.

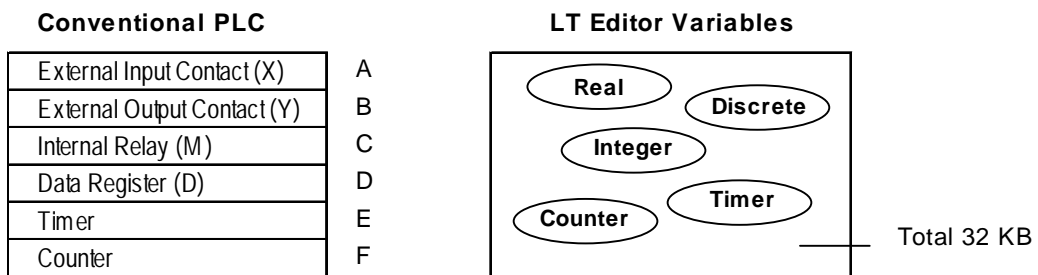
**Reference** For details on defining arrays, refer to 7.3 Access to Variables.

The maximum size of an array (the number of elements it contains) is 65535. However, the actual number of elements that can be used by any application is limited by the size of the LT unit’s variable storage area. The amount of memory available to the LT for variables is limited to 32KB. Be sure to design your system so that the number of variables used does not exceed the LT unit’s memory limit.

Use the following table to find the amount of memory used by each variable.

Variable Type	Memory Used (unit:byte)
Discrete	12
Discrete Array	20 + (No. of elements x 12)
Integer	8
Integer Array	20 + (No. of elements x 8)
Real	16
Real Array	20 + (No. of elements x 16)
Timer	48
Counter	80

In the PLC, the number of variables that can be used by each device is limited. In the LT, however, variables can be registered, regardless of type, as long as the overall limit of 32 KB is not exceeded.



PLC Device and LT Variable Comparison

### ■ Discrete Variables

These variables are used to define a discrete condition, i.e. ON or OFF, using a single bit and the values “0” or “1”.

### ■ Integer Variables

These variables use 32 bits to define integer values from -2147483648 to 2147483647.

### ■ Real Variables

These variables use 64 bits to define floating decimal point values from +/-2.25e<sup>-308</sup> to +/-1.79e<sup>+308</sup>, and “0”.


## ■ Timer/Counter

The Timer and Counter consist of multiple special-purpose variables.

Each dedicated variable's type is set up individually.

### ◆ Timer

The following four dedicated variables are used for Timer instructions.


 *For details, refer to 9.2 Instruction Details.*

Special-Purpose Variables	Description	Variable Type
PT	Preset Value	Integer
ET	Current Value	Integer
Q	Timer Output Bit	Discrete
TI	Timer Measuring Bit	Discrete

\* Any names can be used for these variables.



**Note:** Even when a timer is designated as non-retentive, the special-purpose variable “Timer.PT” will retain data.

 *For a list of retentive/non-retentive variables, refer to n Variable Attributes.*

### ◆ Counter

The following seven dedicated variables are used for the Counter instructions.


 *For details, refer to 9.2 Instructions Details.*

Special Purpose Variables	Description	Variable Type
PV	Preset Value	Integer
CV	Current Value	Integer
R	Counter Reset	Discrete
UP	UP Counter	Discrete
QU	UP Counter Output	Discrete
QD	DOWN Counter Output	Discrete
Q	Counter Output	Discrete

\* Any names can be used for these variables.



- Even when a counter is designated as non-retentive, the special-purpose variable “Counter.PV” will retain data.
- A scan update will not be performed for a counter when it is reset. One scan is required for resetting the counter.

 *For retentive/non-retentive variable details, refer to ■ Variable Attributes.*

### ■ Variable Attributes

Variables have the following attributes, in addition to the variable type.

#### ◆ Internal

Used internally by the LT. It cannot be used for external input/output. Internal variables are equivalent to PLC internal relays (internal registers).

#### ◆ Input/Output

External input/output is available. Assign variables to I/O in the [Configure I/O] window. This feature is equivalent to the input/output relays of the PLC.

**Reference** *For I/O configuration details, refer to 1.9 I/O Configuration.*

#### ◆ Retentive

Retentive-type variables use the LT unit's SRAM, which preserves data values in the case of a power failure. The initial values for these variables are set via Programming mode. When the LT unit is powered down or reset, all current data is retained. However, when the LT unit's Controller is reset in Monitoring mode or by using the #Command, or when logic programs are downloaded, all data is initialized using Programming mode preset values.

In addition, reading the LT unit's .ltx files will save the execution results to the Editor. However, be careful when using retentive-type variables as initial values. If these variables are designed to vary while the logic program is being executed, the predetermined initial values will be lost when the data is loaded into the Editor. Non-retentive variables are either cleared to 0 or set to OFF.

#### ◆ Global

These variables can be designated as either global or non-global. Specify "global" for variables that are used to display Drawing Board Parts. Global variables are automatically registered as LT symbols in the Symbol Editor when you save the ladder logic program. These variables can also be shared with the Drawing Board's display feature. Global/non-global settings of multiple variables can be performed at one time by selecting the desired variables from the Variable List. Up to 2048 global variables can be set.

**Reference** *3.6 Changing Variable Attributes*



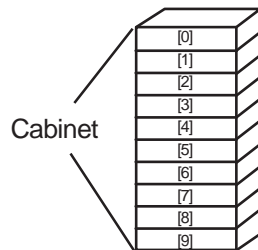
**Note:** Preprogrammed system variables are set to "global" in the LT unit's initial settings.

## 7.3 Accessing Variables

This section explains how to access variable array elements, bits, bytes and words.

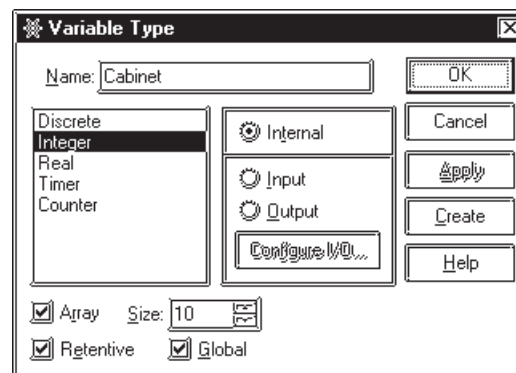
An array is a method of declaring and handling multiple elements with a single variable name. Variables of the same type can be registered as one group using an array.

One analogy is the drawers of a cabinet.



The array variable Cabinet[10] has 10 drawers, numbered from [0] to [9]. These drawers are called Cabinet[0], Cabinet[1], . . . Cabinet[9]. Each drawer corresponds to an individual data register in the PLC.

When using 10 locations of Cabinet memory, first declare the variable name of Cabinet and the array size (number of elements) of 10. The variable type settings are listed as follows:



### ■ Accessing a Discrete Array

To access the elements of a Discrete array, a modifier [n] must be attached to each element. To access the modifier, it is assigned an element number, however the first element number in an array must be “0.”

E.g.: The Discrete array “MotorSetting” is a Discrete array of 10 elements. The seventh element controls the output coil Fan. When the seventh element is turned ON, the output coil turns ON. To access the seventh element of MotorSetting, enter MotorSetting[6].



■ **Accessing an Integer/Integer Array**

Integers and Integer Arrays can be accessed via array elements, bits, bytes, and words.

To access an array’s element, add [n] to the end of the variable name. To access using bits, bytes, and words, the following suffixes are used. The modifier [m] is used to denote the position of the element in the array being accessed.

Access Item/Unit	Suffix
Bit	.X[m]
Byte	.B[m]
Word	.W[m]

◆ **To Access an Element with the Integer Array**

An Integer Array can be used for numerical calculation, tracking of repetitive information and data logging.

E.g.: To record the number of sodas sold in one month in the Integer Array **Water\_Sales**, design your array as follows.

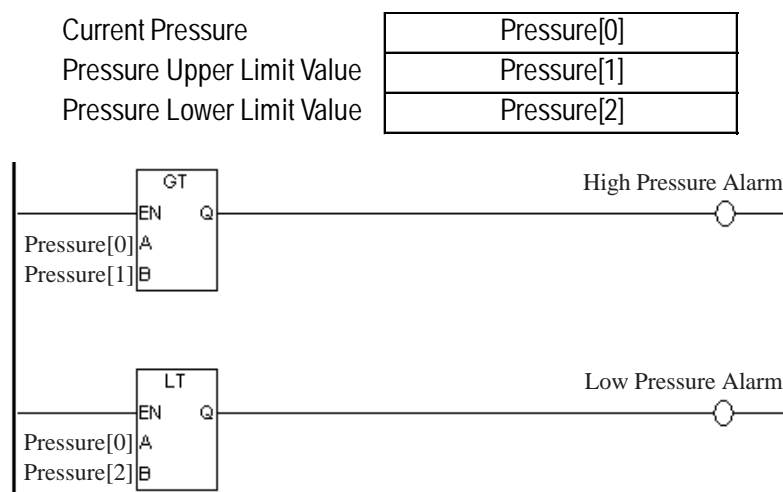
The array consists of 31 Integer type elements which correspond to each of a month’s days (31).

[Day0]	Water_Sales[0]
[Day1]	Water_Sales[1]
[Day2]	Water_Sales[2]
[Day3]	Water_Sales[3]
	—
	—
	—
[Day27]	Water_Sales[27]
[Day28]	Water_Sales[28]
[Day29]	Water_Sales[29]
[Day30]	Water_Sales[30]

The following diagram is an example of the Integer Array **Pressure**, using three elements.

- **Pressure[0]** represents the current pressure of the boiler.
- **Pressure[1]** represents the pressure upper limit value.
- **Pressure[2]** represents the pressure lower limit value.

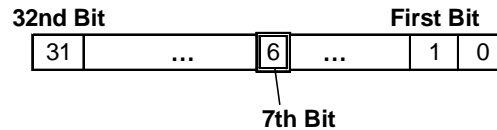
When the pressure is higher or lower than the pressure limit, an alarm turns ON.



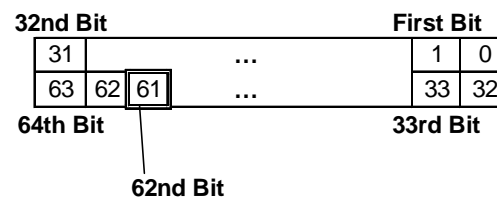
◆ **Accessing an Integer Array using bits**

As is the case with the discrete array variables, integer arrays can be accessed via bits, bytes, and words. To access the **m+1st** bit of the **n+1st** element in the **Integer\_Array\_Variable\_Drink\_Sales**, enter **Drink\_Sales[n].X[m]**.

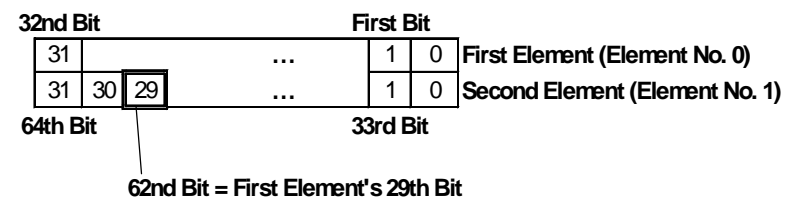
E.g.: • To access the Integer array **Alarm**'s seventh bit, type **Alarm.X[6]**.



- To access the 62nd bit of the Integer array variable **Water\_Sales**, type **Water\_Sales.X[61]**.



Also, for **Water\_Sales[1].X[29]**:



As a result, since **Water\_Sales.X[61] = Water\_Sales[1].X[29]**, both can be used to access the 62nd bit of the Integer array **Water\_Sales**.

- To access the 6th byte of the Integer array variable **Water\_Sales**, both **Water\_Sales.B[5]** and **Water\_Sales[1].B[1]** can be used.
- To access the 5th word of the Integer array variable **Water\_Sales**, both **Water\_Sales.W[4]** and **Water\_Sales[2].W[0]** can be used.



**Note:** **Water\_Sales.X[61]** and **Water\_Sales[0].X[61]** have the same meaning.

In the following example the 3rd bit of the system variable **#Status** is used as a NO instruction variable. The third bit of **#Status** identifies whether the LT unit has an I/O error or not. Therefore, when the third bit is turned ON, the output coil's **IO\_Error** is turned ON, which provides notification that an I/O error has occurred.



■ **Accessing a Real Array**

Real Arrays can be accessed using array elements. To access the elements of a Real array, the modifier (n) must be attached to each element, which represents the element number. Also, “0” is used for the first element in the array.

E.g.: To access the 5th element in the Real array **SolutionTemperature**, you would use "**SolutionTemperature[4]**".



**Note:** The LT Editor can handle up to 2048 LT variables. The elements of the array become single variables. Thus, an array with five elements becomes five variables.

An Real array can be used for numerical calculation, tracking of repetitive information and data logging.

E.g.: To record the temperature of a solution every 24 hours in the Real array **Solution\_Temperature**, the structure of data is as follows.

The array consists of 24 Real type elements that correspond to each hour of a day.

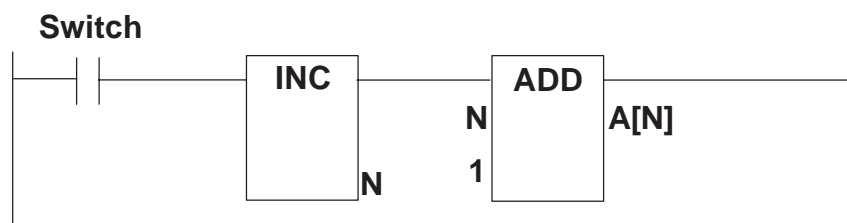
Real element 0 corresponds to the temperature data at 0:00.

Solution_Temperature[0]
Solution_Temperature[1]
Solution_Temperature[2]
Solution_Temperature[3]
—
—
—
Solution_Temperature[20]
Solution_Temperature[21]
Solution_Temperature[22]
Solution_Temperature[23]

■ **Array Indirect Access**

Array elements [n] can be indirectly accessed by an Integer variable. Numbers in the square brackets [ ] of suffixes such as .X[m], B[m], and W[m] can also be indirectly accessed.

The following example assumes that you press the switch. In the INC instruction, "N" increments by one with every single scan. The result of the ADD instruction, i.e. the sum of "N" and "1", is then assigned to A[N]. After 5 scans have been performed, "1" is assigned to A[0], "2" to A[1], "3" to A[2], "4" to A[3], and "5" to A[4]. Note that the initial value of "N" is 0.



# 8 System Variables

The following table provides a list of the Controller's predefined System Variables.

## 8.1 System Variable List

System Variables are used to display the Controller's current condition, and effect its operation. System variables have similar variable types as ordinary variables and perform similarly. Since system variables are preprogrammed and defined, they cannot be deleted and their names cannot be changed.

Group	System Variable	Explanation	Initial Value	Variable Name	
Data	#AvgLogicTime	Displays the average Logic Time (Read, Perform, Write) once every 64 scans. (Unit ms)	0	Integer	Read Only
	#AvgScanTime	Displays the latest Logic Time (Read, Perform, Write, Display processing). (Unit ms)	0	Integer	
	#Clock100ms	Create 0.1s clock.	-	Discrete	
	#Day	Stores Day data as BCD two digits.	-	Integer	
	#EditCount	Currently not used by LT	-	Integer	
	#ForceCount	Counts the number of times a variable is forced ON or OFF.	0	Integer	
	#IOStatus	Displays the I/O Driver's condition.	-	Integer [10]	
	#LogicTime	Displays the latest Logic Scan Time (Read, Perform, Write). (Unit ms)	0	Integer	
	#Month	Stores Month data as BCD two digits.	-	Integer	
	#Platform	Indicates the controller's platform.	-	Integer	
	#ScanCount	Excluding the current scan, counts the number of scans performed.	0	Integer	
	#ScanTime	Displays the latest Logic Scan Time (Read, Perform, Write, Display processing). (Unitms)	0	Integer	
	#Status	Indicates the controller's current status.	-	Integer	
	#StopPending	Currently not used by LT	-	Discrete	
	#Time	Stores Time data as BCD four digits.	-	Integer	
	#Version	Displays the controller's version data.	-	Integer	
	#WCLScan	Currently not used by LT	-	Integer	
	#WCLStatus	Currently not used by LT	-	Integer	
	#WeekDay	Stores Weekday data as value of 0 to 6	-	Integer	
	#Year	Stores Year data as BCD two digits.	-	Integer	
#WatchdogTime	Displays the value set via the Editor or the GP unit's OFFLINE screen. (Unitmsec.)	-	Integer		



Group	System Variable	Explanation	Initial Value	Variable Name	
Errors	#FaultCode	Displays the latest error code.	-	Integer	Read Only
	#FaultRung	Displays the rung where the error occurred.	-	Integer	
	#IOFault	Turns ON when an error occurs.	-	Discrete	
	#Overflow	Turns ON when an overflow occurs due to mathematical commands or conversion of a variable from Real to Integer.	0	Discrete	
Settings	#Command	Changes the controller's mode.	0	Integer	Write Only
	#DisableAutoStart	Defines the mode entered when the GLC starts up.	-	Discrete	
	#Fault	Used to stop the performance of an Error Handler subroutine.	0	Discrete	
	#FaultOnMinor	Controls the completion of the logic performed when a minor error occurs.	0	Discrete	
	#PercentAlloc	Calculates the Percent Scan's percentage. (Unit: %)	0	Integer	
	#PercentMemCheck	Not currently used by the LT	-	Integer	
	#Screen	Switches LT screens by assigning screen numbers.	-	Integer	
	#StopScans	Not currently used by the LT	-	Integer	
	#TargetScan	Sets the Constant Scan Time. (Unit: ms)	-	Integer	



**#Year, #Month, and #Day are saved as the LT unit's time data. Time data changes are performed via the LT unit's Initial settings, or the System Data Area's Write settings.**

**Reference** *LT Series User Manual (sold separately), External Device Connection Manual.*

**8.1.1 How to Use System Variables**

This section uses a #Screen to explain how to use system variables.

The following logic program switches the screen to base screen (B100), which is screen number 100. Pressing the switch changes the screen by substituting 100 in the #Screen.



## 8.2 System Variable Details

This section describes each system variable in detail.

### 8.2.1 #AvgLogicTime

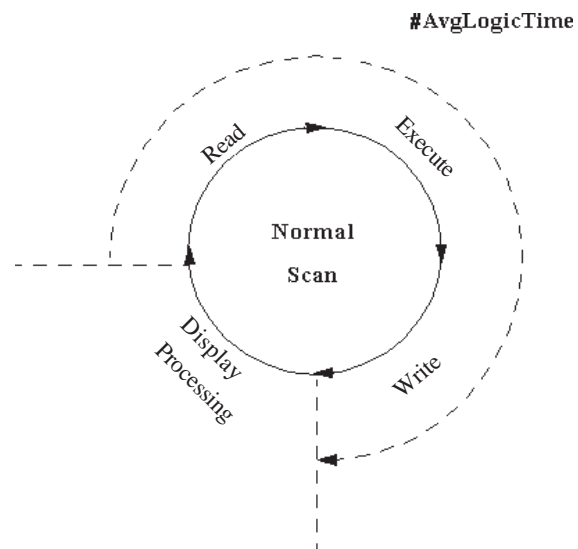
#AvgLogicTime stores the average logic time in ms units.

The average logic time refers to the average time required in one scan for reading I/O, executing the ladder logic program, and reading I/O. Every 64 scans, this system variable updates the average logic time since its last calculation.

Variable Type: Integer

Set by: Controller

Read Only



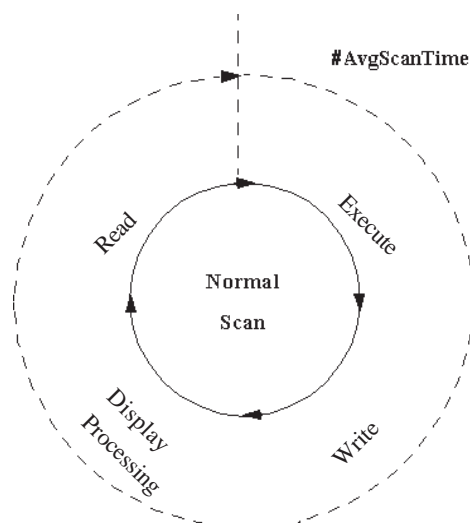
### 8.2.2 #AvgScanTime

#AvgScanTime stores the average amount of time, in milliseconds, that the controller uses to read inputs, execute logic, write outputs, and perform display processing in a single scan. Every 64 scans, this system variable updates the average scan time.

Variable Type: Integer

Set by: Controller

Read Only



**8.2.3 #Clock100ms**

#Clock100ms generates clock in milliseconds. Do not change the clock value since this is used for read in only. An initial value is undefined.

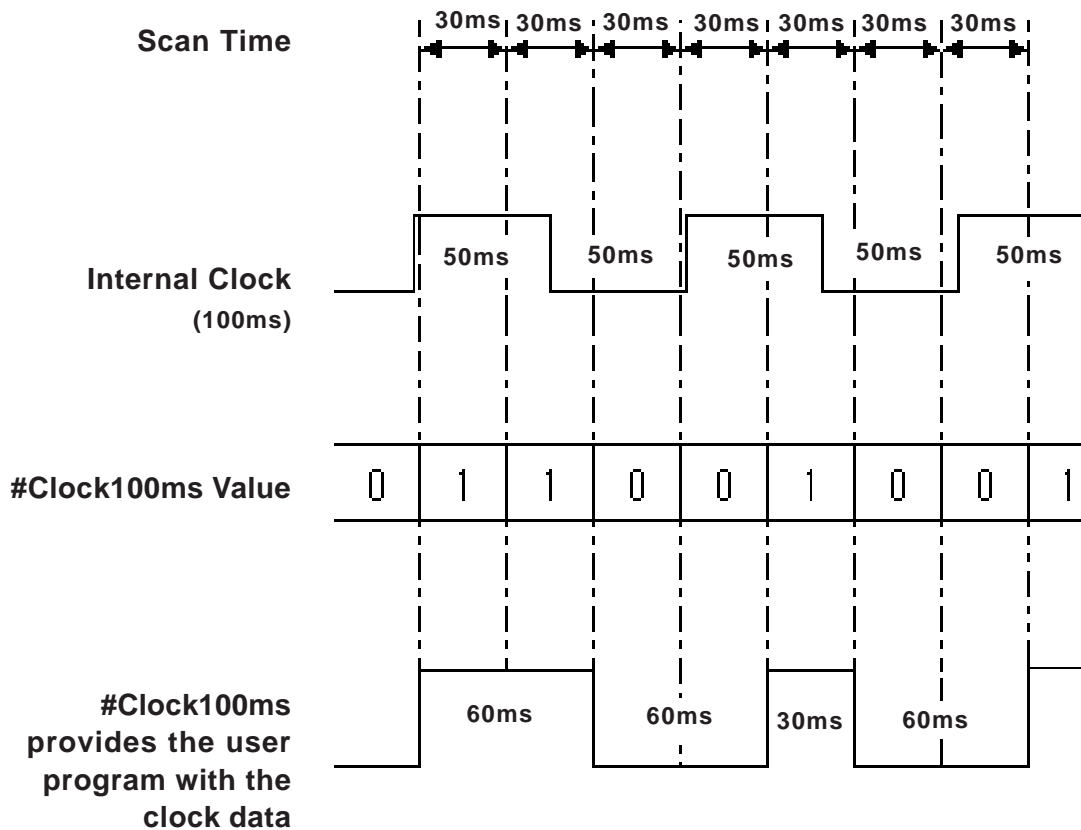
Variable Type: Discrete  
 Set by: Controller  
 Read Only



**Note:**

- If an LT unit's scan time exceeds 50ms, #Clock100ms clock will not be guaranteed.
- If the #Clock100ms clock reads in the internal clock 100ms at the beginning of each LT scan, an error will occur.

**Scan Time Every 30ms**



**#Clock100ms includes approximately the same amount of error as the scan time.**

### 8.2.4 #Day

#Day displays the Day data, as set by the controller, using two digits in BCD format.

Variable Type: Integer

Set by: Controller

Read Only

Year, Month, Day, and Time data are displayed using the following system variables:

E.g., July 14, 2001 at 6:19 a.m.

	Year	Month	Day	Time
<b>System Variable</b>	#Year	#Month	#Day	#Time
<b>Value</b>	01	07	14	0619

### 8.2.5 #ForceCount

#ForceCount stores the number of variables that are forced ON or OFF in the current ladder program.

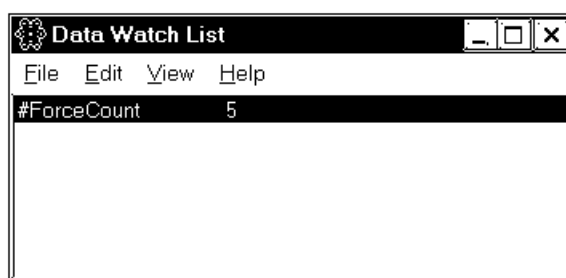
**Reference** Refer to 2.2 Starting and Stopping the Controller.

Variable Type: Integer

Set by: Controller

Read Only

The Data Watch List window indicates the five variables that are forced ON or OFF in the logic program.



**8.2.6 #IOStatus**

#Day #IOStatus is set by the I/O driver, and stores the I/O driver’s current status in #IOStatus[1].

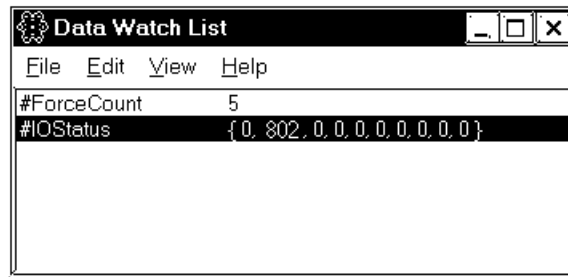
A value of 0 indicates that the I/O is normal. The status indicated by a value other than 0 differs, depending on the I/O driver.

Variable Type: Integer[10]

Set by: Controller

Read Only

The Data Watch List window shows that Error 802 occurred in the I/O driver 1.



**Reference** For I/O driver error code descriptions, refer to *Chapter 11 I/O Drivers*.

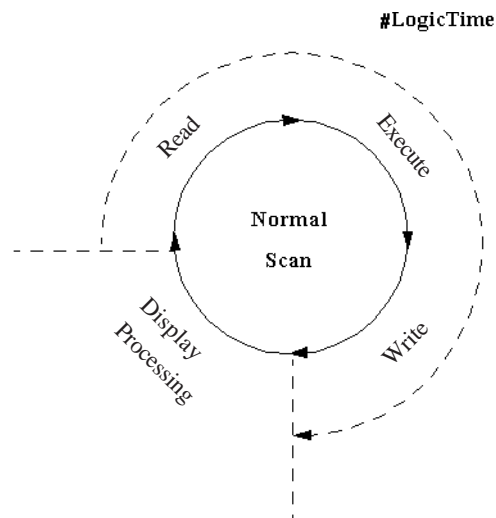
**8.2.7 #LogicTime**

#LogicTime indicates the amount of time, in milliseconds, that the controller uses in a single scan to read inputs, execute logic, and write outputs of the previous scan. Logic time does not include the display processing time allowed by the controller for other programs to execute.

Variable Type: Integer

Set by: Controller

Read Only



**8.2.8 #Month**

#Month displays the Month data, as set in the controller, using two digits in BCD format.

Variable Type: Integer

Set by: Controller

Read Only

Year, Month, Day, and Time data are displayed using the following system variables:

E.g., July 14, 2001 at 6:19 a.m.

	Year	Month	Day	Time
<b>System Variable</b>	#Year	#Month	#Day	#Time
<b>Value</b>	01	07	14	0619

**8.2.9 #Platform**

#Platform displays which platform the controller is running on.

Variable Type: Integer

Set by: Controller

Initial Value: 1

Read Only

Value	Platform
16#54	LT Type A
16#64	LT Type B/Type B+
16#74	LT Type C
16#114	LT Type H-AD
16#144	LT Type H-ADP
16#154	LT Type H-ADT

**8.2.10 #ScanCount**

#ScanCount is a counter incremented by the controller at the end of each scan.

The value range of #ScanCount is 0 – 16#FFFFFFFF. When the counter value exceeds the maximum value (16#FFFFFFFF), the value of #ScanCount is set to zero (functioning as a Rollover, but without setting the Overflow variable).

Variable Type: Integer

Set by: Controller

Read Only



**Note:** Whether or not the logic program is running can be easily checked using #ScanCount.

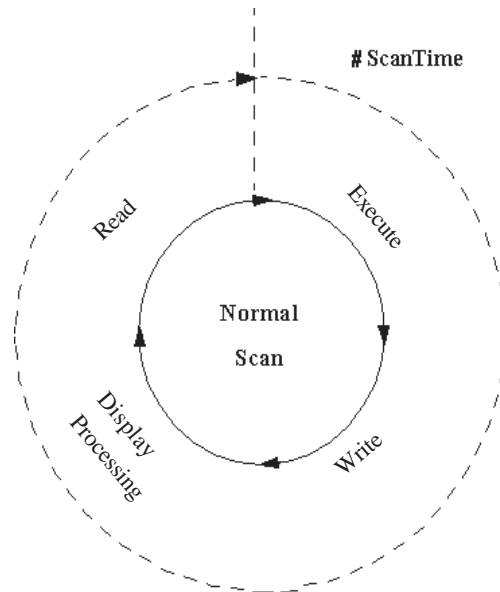
### 8.2.11 #ScanTime

#ScanTime stores the amount of time, in milliseconds, that the controller uses during its last complete scan, to read I/O, execute logic, write I/O, and display processing.

Variable Type: Integer

Set by: Controller

Read Only



### 8.2.12 #Status

#Status indicates the controller's status.

Within the #Status system variable:

Byte 0 indicates the current fault conditions of the controller.

Byte 1 is used to show the fault status history, and is reset to 0 only when the controller is reset.

Byte 2 indicates the current operating status of the controller.

Byte 3 is reserved.

Variable Type: Integer

Set by: Controller

Read Only



**Note:**

Intermittent errors can be detected by using the latch fault flag. Use hexadecimal format for #Status.

When the following fault flags become 1, the corresponding conditions are indicated as follows:

		Fault Flags
<b>Byte0</b>	Bit0	Major fault
	Bit1	Minor fault
	Bit2	I/O fault
	Bit3	Reserved
	Bit4	Read error
	Bit5	Reserved
	Bit6	Scan time error
	Bit7	Reserved

		Latched Fault Flags
<b>Byte1</b>	Bit8	Major fault
	Bit9	Minor fault
	Bit10	I/O fault
	Bit11	Reserved
	Bit12	Read error
	Bit13	Reserved
	Bit14	Scan time error
	Bit15	Reserved

		Controller Status
<b>Byte2</b>	Bit16	Running
	Bit17	I/O Enabled/Disabled
	Bit18	Forces Enabled/Disabled
	Bit19	Paused
	Bit20	Reserved
	Bit 21-23	Reserved

<b>Byte3</b>	Reserved
--------------	----------

**8.2.13 #Time**

#Time displays Time data, as set in the controller, using four digits in BCD format.

Variable Type: Integer

Set by: Controller

Read Only

Year, Month, Day, and Time data are displayed using the following system variables:

E.g., July 14, 2001 at 6:19 a.m.

	Year	Month	Day	Time
<b>System Variable</b>	#Year	#Month	#Day	#Time
<b>Value</b>	01	07	14	0619



**8.2.14 #Version**

#Version indicates the version number of the controller. #Version is displayed in hexadecimal format.

Variable Type: Integer

Set by: Controller

Read Only

Byte No.	Description	Ver. 1.0.0
Byte3	Major version	01
Byte2	Minor version	00
Byte1	Reserved	-
Byte0	Reserved	-

**8.2.15 #Year**

#Year displays Year data, as set in the controller, using two digits in BCD format.

Variable Type: Integer

Set by: Controller

Read Only

Year, Month, Day, and Time data are displayed using the following system variables:

E.g., July 14, 2001 at 6:19 a.m.

	Year	Month	Day	Time
<b>System Variable</b>	#Year	#Month	#Day	#Time
<b>Value</b>	01	07	14	0619

**8.2.16 #Weekday**

#Weekday displays present Weekday data, with a value of 0 to 6.

Variable Type: Integer

Set by: Controller

Read Only

#Weekday reflects the number LS2054.

LS2054 cycles a value of 0 to 6 (.. 5 6 0 1) at the time-of-day change (23:59 to 00:00).

When the power is plugged in, values of 0 to 6 are reflected by default. It is not necessarily reflect that Sunday is 0 and Monday is 1. To coordinate Weekday data with the actual day of the week, map values of 0 to 6 to LS2062.

**8.2.17 #FaultCode**

#FaultCode identifies the most recent fault status. A controller resets all these values to 0.

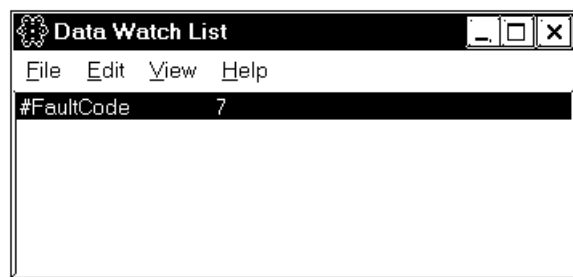
**Reference** Refer to 12.2 Error Codes.

Variable Type: Integer

Set by: Controller

Read Only

Code	Type	Cause
0	Normal	No fault.
1	Minor	Overflow resulting from a mathematical operation or a Real-to-Integer conversion.
2	Major	Array reference is out of bounds.
3	Major	Bit reference of the Integer (32 bits) is out of bounds.
4	Major	Stack overflow.
5	Major	Invalid instruction code.
6		Reserved by the system
7	Major	Scan time exceeds watchdog time.
8		Reserved by the system
9	Major	Software error – typically a malfunctioning custom function block – may require a system reboot to recover.
10		Reserved by the system
11		Reserved by the system
12	Minor	BCD/BIN conversion error
13	Minor	ENCO/DECO conversion error
14		Reserved by the system



In the Data Watch List window, #FaultCode 7 is displayed.

This indicates that the scan time has exceeded the watchdog time.

**8.2.18 #FaultRung**

#FaultRung stores the rung number where a fault occurred. #FaultRung is set to 0 if there are no faults.

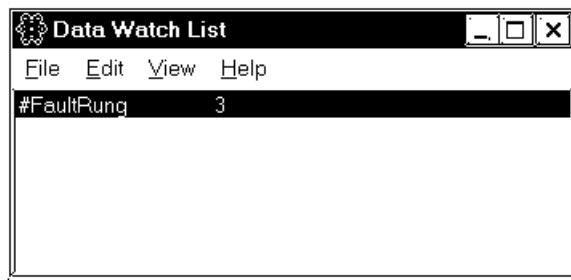
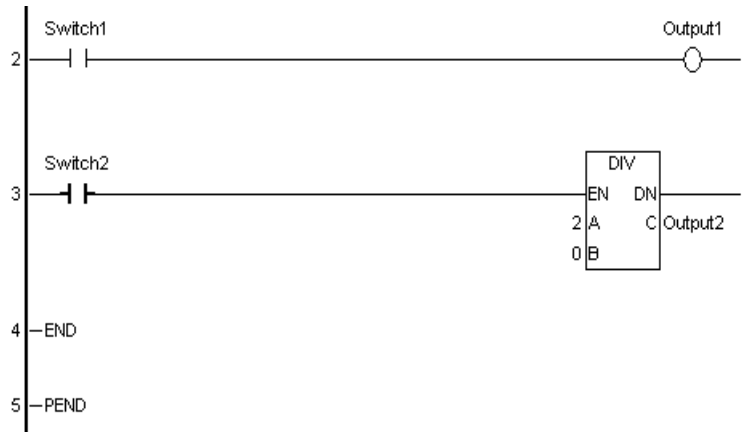
The following example shows when an error occurred at Rung 3.

This error is caused by subtracting the Integer by 0 when DIV Instruction is executed. This error remains until the next error occurs or the controller is reset.

Variable Type: Integer

Set by: Controller

Read Only



**8.2.19 #IOFault**

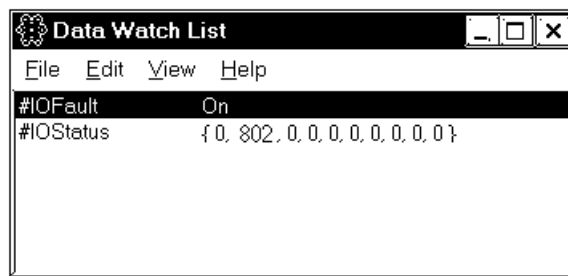
#IOFault turns ON when an I/O fault occurs with the I/O driver. This error remains until the next error occurs or the controller is reset. Check the #IOStatus variable for detailed status of the I/O driver.

When #IOFault turns ON, #IOFault is displayed in the Data Watch List window.

Variable Type: Discrete

Set by: Controller

Read Only



**Reference** For I/O driver error code descriptions, refer to **Chapter 11 I/O Drivers**.

**8.2.20 #Overflow**

#Overflow turns ON when a mathematical fault occurs. #Overflow stays ON until the next mathematical instruction or conversion.

Mathematical faults include instruction overflows, Real-to-Integer conversion overflows, and divide by zero errors.

When a mathematical fault occurs, a minor fault also occurs, which executes an ErrorHandler subroutine, if one exists.

▼ **Reference** ▲ Refer to 12.2 Error Codes.

The ErrorHandler subroutine is an error process subroutine, and must first be created under the name “ErrorHandler.”

The value in the #Fault system variable defined whether the controller will stop or continue execution of the logic program.

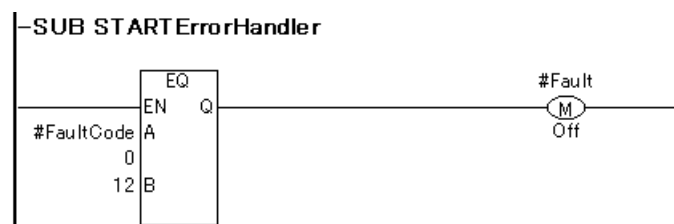
▼ **Reference** ▲ Refer to 8.2.21 #Fault.

Variable Type: Discrete

Set by: Controller

Read Only

In the following example, the ErrorHandler subroutine detects BCD/BIN conversion errors and stops execution of the logic program.



**Note:** If an overflow does not occur during Real-to-Integer conversion, #Overflow will not turn ON.

**8.2.21 #Command**

#Command is an Integer variable used as a controller command. After the controller reads #Command, it resets the value to 0. When multiple bits are ON, the lowest bit takes precedence.

Variable Type: Integer

Set by: User

Initial Value: OFF (All bits)

Writable

Bit0 (=1)	Stop Controller
Bit1 (=2)	Run Controller
Bit2 (=4)	Reset Controller
Bit3 (=8)	Execute single scan
Bit4 (=16)	Continue
Bit5 (=32)	Pause
Bit7 (=128)	Enable I/O

**8.2.22 #DisableAutoStart**

If the power is turned ON while #DisableAutoStart is ON, the controller starts up in the STOP mode.

If the power is turned ON while #DisableAutoStart is OFF, the controller starts up in the state it was in (START or STOP) prior to shutdown.

The above settings are enabled only when the Controller State setting is set to Default in the LT unit's initial settings.

Variable Type: Discrete

Set by: User

Initial Value: OFF

Writable

**8.2.23 #Fault**

#Fault is referred to by the controller as to whether the logic program will stop or continue to execute at the completion of the ErrorHandler subroutine.

By turning #Fault ON, the controller will be able to stop executing the logic program.

**Reference** For information about ErrorHandler subroutines, refer to 8.2.18 #Overflow.

Variable Type: Discrete

Set by: User

Initial Value: OFF

Writable



**Note:** #Fault has no meaning when there is no ErrorHandler subroutine.

**8.2.24 #FaultOnMinor**

#FaultOnMinor is checked by the controller to determine whether the logic program will stop or continue to execute when a minor fault occurs and there is no ErrorHandler subroutine in the logic program.

Turning ON the #FaultOnMinor allows you to pause the execution of a ladder logic program.

**▼Reference▲** Refer to *12.2 Error Codes*.

*For information about ErrorHandler subroutine, refer to 8.2.18 #Overflow.*

Variable Type: Discrete

Set by: User

Initial Value: OFF

Writable

**8.2.25 #PercentAlloc**

#PercentAlloc is used when the controller is set to the Percent Scan mode. It sets the percentage of the LT unit's total CPU time available to the controller. Set a scan time value in multiples of 10ms.

#PercentAlloc can be set in the initial settings or the configuration settings when the controller is in RUN mode. Usually, #PercentAlloc can be set up in the Setup dialog box.

**▼Reference▲** Refer to *6.1.2 RUN Mode*.

Variable Type: Integer

Set by: User

Range: 0 to 50%

Initial Value: 50

Writable

**8.2.26 #Screen**

#Screen is used to change LT unit screens. This screen change variable’s operation differs from [Change Screen Check] as follows.

If the [Change Screen Check] feature is enabled and the screen change number is entered in #Screen, after the screen change is completed the value is reset to “0”.

**Reference** *Ch. 1 Programming*

Variable Type: Integer

Set by: User/Controller

Initial Value: 0

Writable



- The screen number set in #Screen defines which base screen to display. This number is not the currently displayed screen number.
- Since the #Screen variable is write only, it cannot be used to determine if a screen has changed, etc.



- **When changing screens, use the #Screen in the logic program. Do NOT write directly to the #Screen using touch input. Change screens using the logic program diagram below as an example.**



- **After power is turned ON, if the #Screen variable is used to change the initial screen, be sure to wait more than 200ms or use the LSS[0].x[3] (LS2032’s bit 3) bit rising (0 -> 1) timing.**

**8.2.27 #TargetScan**

#TargetScan is used when the controller is set to the Constant Scan mode.

The #TargetScan variable is designated in multiples of 10ms units.

When the logic time is constant, increasing the value in #TargetScan means that the display processing time will be longer.

Decreasing the value in #TargetScan means that the display processing time will be shorter. This is because most of the processing time is used by the controller.

#TargetScan can be set in the initial settings or the configuration settings when the controller is in RUN mode. Typically, #TargetScan can be set up in the Setup dialog box.

**▼Reference▲** Refer to **6.1.2 RUN Mode**.

Variable Type: Integer

Set by: User

Range: 10–2000ms

Initial Value: 10ms

Writable

**8.2.28 #WatchdogTime**

#WatchdogTime is used to set the value of the watchdog timer, in milliseconds. When #ScanTime exceeds this value, a major fault occurs.

**▼Reference▲** Refer to **12.2 Error Codes**.

The #WatchdogTime setting should not be changed while the controller is starting up.

Even if the value is changed, the actual setting will not change. Usually, #WatchdogTime is set in the OFFLINE mode's [Setup] dialog box.

Variable Type: Integer

Set by: User (prior to starting RUN mode)

Initial Value: 500ms

Read-only



# 9 Instructions

## 9.1 Instruction List

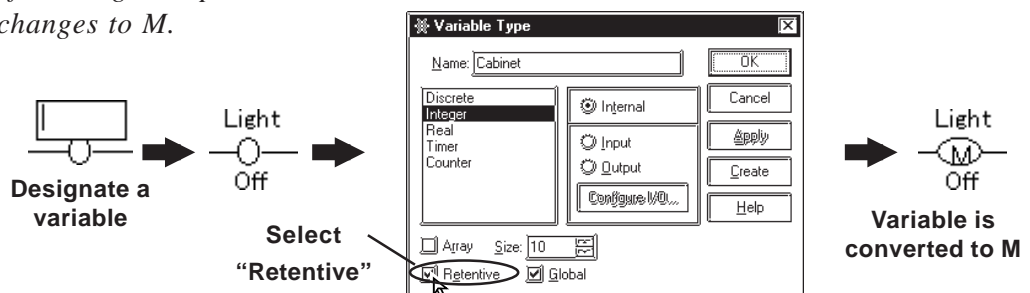
The Instructions supported by the LT Editor software are as follows.

### ■ Discrete Instructions

Instruction	Type	Symbol	Function
NO	Normally Open		Allows power to pass when the contact turns ON.
NC	Normally Closed		Allows power to pass when the contact turns OFF.
OUT/M <sup>*1</sup>	Output Coil/Retention Coil		Turns physical output devices or internal discrete variables and expressions ON or OFF.
NEG/NM <sup>*1</sup>	Negated Coil/Negated Retention Coil		Turns a variable OFF if the coil receives power, and ON if it doesn't.
SET/SM <sup>*1</sup>	Latch Coil/ Latch Retention Coil		Turns a variable ON if the coil receives power. Stays ON until receiving another explicit instruction.
RST/RM <sup>*1</sup>	Unlatch Coil/ Unlatch Retention Coil		Turns a variable OFF if the coil receives power. Stays OFF until receiving another explicit instruction.
PT <sup>*2</sup>	Positive Transition		Allows power to pass if the variable was OFF during the previous scan, but is ON now.
NT <sup>*2</sup>	Negative Transition		Allows power to pass if the variable was ON during the previous scan, but is OFF now.

- For the instructions listed above, when a variable is retentive, it automatically changes to one of the right-side instructions. Therefore, when entering data in this screen, be sure to use one of the left-side (non-retentive) instructions.

In the following example, when an OUT instruction's variable is retentive, the screen icon changes to M.



- Up to 2048 PT/NT instructions can be used.

■ Arithmetic Operation Instructions

Instruction	Type	Symbol	Function
AND	Logical Multiply		A and B → C Normal Continuity
OR	Logical Add		A or B → C Normal Continuity
XOR	Exclusive Logical Add		A xor B → C Normal Continuity
NOT	Bit Negation		$\bar{A} \rightarrow C$ Normal Continuity

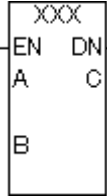

■ Movement Instructions

Instruction	Type	Symbol	Function
MOV	Transfer		IN → OUT Normal Continuity
BMOV	Block Transfer		
FMOV	File Transfer		


■ Shift Instructions

Instruction	Type	Symbol	Function
ROL	Rotate Left		
ROR	Rotate Right		
SHL	Shift Left		
SHR	Shift Right		


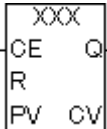
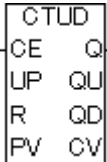
■ **Mathematical Instructions**

Instruction	Type	Symbol	Function
ADD	Add		$A + B \rightarrow C$ Normal Continuity
SUB	Subtract		$A - B \rightarrow C$ Normal Continuity
MUL	Multiply		$A \times B \rightarrow C$ Normal Continuity
DIV	Divide		$A \div B \rightarrow C$ Normal Continuity
MOD	Residual Processing		$A \% B \rightarrow C$ Normal Continuity
INC	Increment		$A + 1 \rightarrow A$ Normal Continuity
DEC	Decrement		$A - 1 \rightarrow A$ Normal Continuity

■ **Comparison Instructions**

Instruction	Type	Symbol	Function
EQ	Equal To (=)		When $A = B$ , Continuity
GT	Greater Than (>)		When $A > B$ , Continuity
LT	Less Than (<)		When $A < B$ , Continuity
GE	Greater Than or Equal To ( $\geq$ )		When $A > \text{ or } = B$ , Continuity
LE	Less Than or Equal To ( $\leq$ )		When $A < \text{ or } = B$ , Continuity
NE	Not Equal (< >)		When $A < > B$ , Continuity

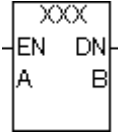
■ **Timer and Counter Instructions**

Instruction	Type	Symbol	Function
TON	ON Delay Timer		See 4.2.33 – "TON (ON Delay Timer)."
TOF	OFF Delay Timer		See 4.2.34 – "TOF (OFF Delay Timer)."
TP	Timer Pulse		See 4.2.35 – "TP (Timer Pulse)."
CTU	UP Counter		See 4.2.36 – "CTU (UP Counter)."
CTD	DOWN Counter		See 4.2.37 – "CTD (DOWN Counter)."
CTUD	UP/DOWN Counter		See 4.2.38 – "CTUD (UP/DOWN Counter)."

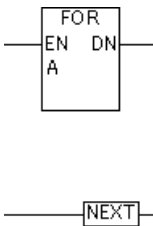


**The Timer Instruction includes an approximate amount of error equal to the scan time.**

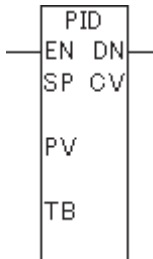
■ Convert Instructions

Instruction	Type	Symbol	Function
BCD	BCD Conversion		A → BCD conversion → B Normal Continuity
BIN	Binary Conversion		A → Binary conversion → B Normal Continuity
ENCO	Encode		A → Encode conversion → B Normal Continuity
DECO	Decode		A → Decode conversion → B Normal Continuity

■ Program Control Instructions

Instruction	Type	Symbol	Function
JMP	Jump	->>label name	Jumps to a label
JSR	Jump to Subroutine	->>Subroutine Name<<-	Jumps to subroutine
RET	Return from Subroutine	-<RETURN>-	Returns to called JSR command.
FOR, NEXT	Repeat		Repeats execution of the logic program between FOR and NEXT for the number of times assigned at A.

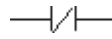
■ Special Instruction \*1

Instruction	Type	Symbol	Function
PID	PID Calculation		<p><u>When EN is energized:</u> SP and PV perform the PID calculation, and output via CV.</p> <p><u>When EN is not energized:</u> TB and CV go to MOV.</p>

## 9.2 Instruction Details

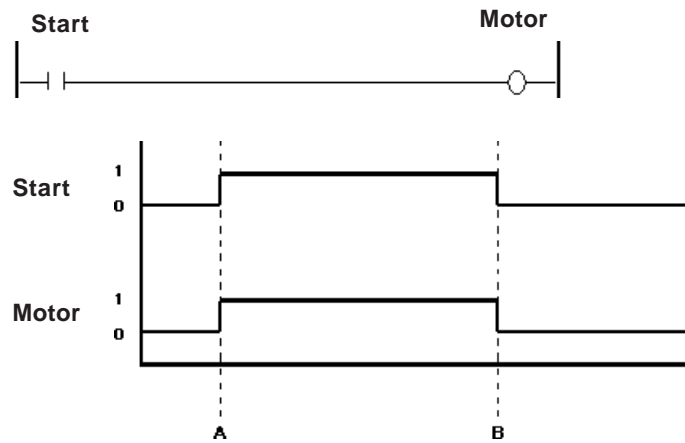
### 9.2.1 NO (Normally Open)

Variable



The NO instruction allows power to pass when the variable is ON.

The following diagram is an example of the NO instruction's function.



A: When the Start variable turns ON, the Motor variable turns ON.

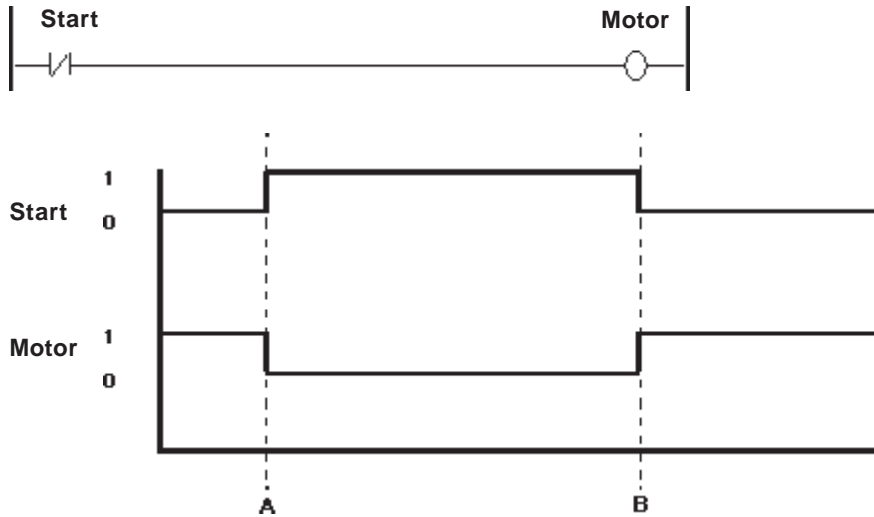
B: When the Start variable turns OFF, the Motor variable turns OFF.

**9.2.2 NC (Normally Closed)**

Variable



The NC instruction allows power to pass when the variable is OFF.  
 The following diagram is an example of the NC instruction's function.

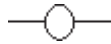


A: When the Start variable turns ON, the Motor variable turns OFF.

B: When the Start variable turns OFF, the Motor variable turns ON.

### 9.2.3 OUT/M (Output Coil)

Variable



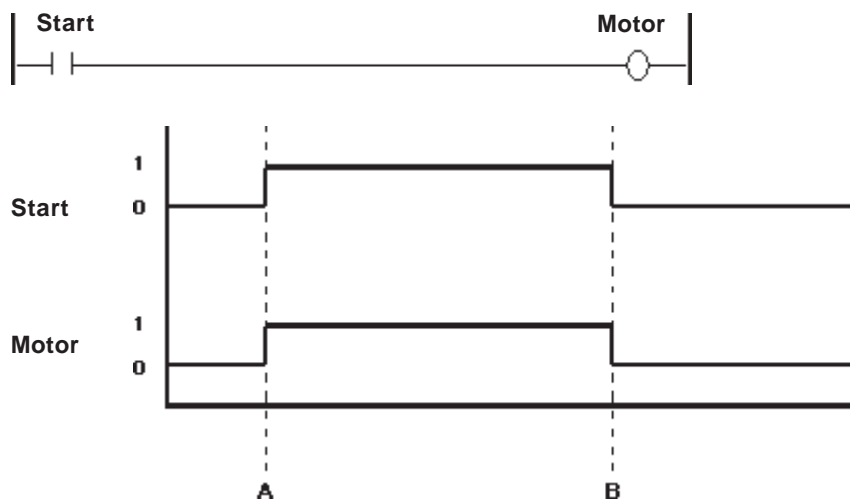
The OUT instruction is used to turn ON/OFF the variables mapped to the I/O, or the Discrete variables in the internal memory .

Since this instruction is a coil-type output instruction, only one instruction can be used for each rung. Other instructions cannot be used on the right side of the output instruction. The output instruction should be placed immediately left of the right-hand power line.

When the variable mapped to the OUT instruction is retentive, the following symbol is displayed in the logic program.



The following diagram is an example of the OUT instruction's function .



A: When the Start variable turns ON, the Motor variable turns ON.

B: When the Start variable turns OFF, the Motor variable turns OFF.



**Note:**

The OUT instruction can be used only with non-retentive variables. With retentive variables, use the M (Retention Coil) instruction.

**9.2.4 NEG (Negated Coil)**

Variable



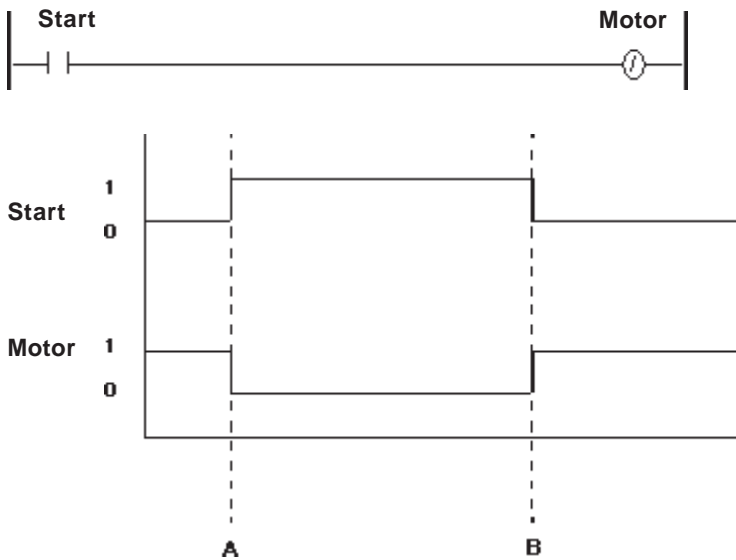
When the NEG instruction is executed, the variable turns OFF when the coil receives power, and ON when the coil does not receive power.

Since this instruction is a coil-type output instruction, only one instruction can be used for each rung. Other instructions cannot be used on the right side of the output instruction. The output instruction should be placed immediately left of the right-hand power line.

When the variable mapped to NEG instruction is retentive, the following symbol is displayed in the logic program.



The following diagram is an example of the NEG instruction's function.



A When the Start variable turns ON, the Motor variable turns OFF.

B When the Start variable turns OFF, the Motor variable turns ON.



**Note:** The NEG instruction can be used only with non-retentive variables.



## 9.2.5 SET (Set Coil)

Variable



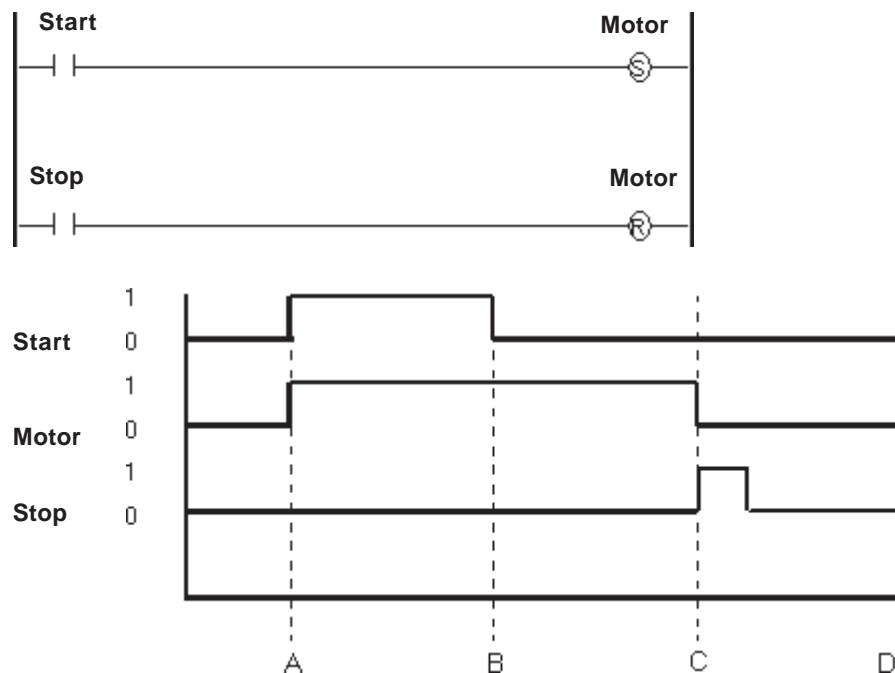
When the SET instruction is executed after the coil receives power, the variable turns ON. The variable will remain ON until explicitly turned OFF by another instruction (such as an RST instruction).

Since this instruction is a coil-type output instruction, only one instruction can be used for each rung. Other instructions cannot be used on the right side of the output instruction. The output instruction should be placed immediately left of the right-hand power line.

When the variable mapped to SET instruction is retentive, the following symbol is displayed in the logic program.



The following diagram is an example of the SET instruction's function.



A: When the Start variable turns ON, the Motor variable turns ON.

B: The Start variable turns OFF, but does not affect the Motor variable.

C: The Stop variable turns ON, the Motor variable resets.

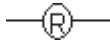
D: The Motor variable stays reset until the Start variable turns ON.



**Note:** The SET instruction can be used only with non-retentive variables. With retentive variables, use the SM (Latch Retention Coil) instruction.

**9.2.6 RST (Reset Coil)**

Variable



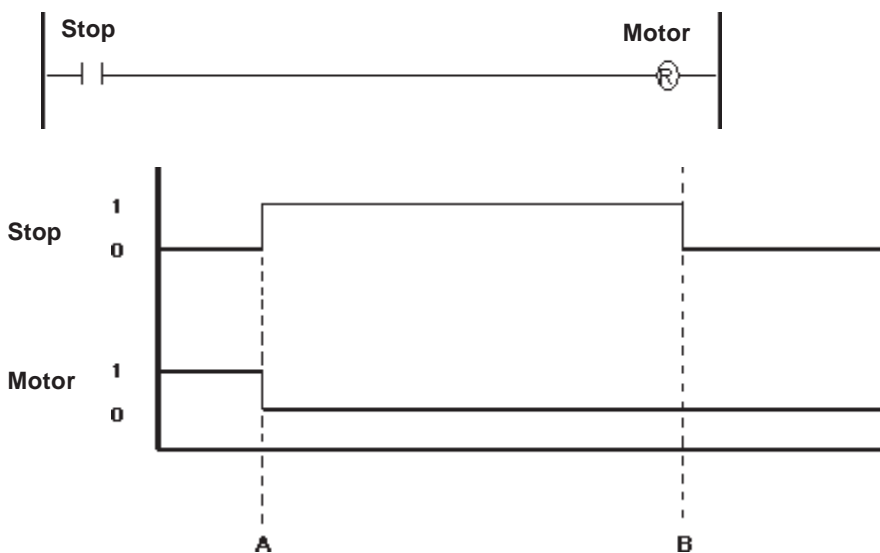
When the coil receives power after the RST instruction is executed, the variable turns OFF. The variable remains OFF until explicitly turned ON by another instruction (such as a SET instruction).

Since this instruction is a coil-type output instruction, only one instruction can be used for each rung. Other instructions cannot be used on the right side of the output instruction. The output instruction should be placed immediately left of the right-hand power line.

When the variable mapped to the RST instruction is retentive, the following symbol is displayed in the logic program.



The following diagram is an example of the RST instruction's function.



A: When the Stop variable turns ON, the Motor variable resets.

B: When the Stop variable turns OFF, the Motor variable reset by the RST instruction will remain OFF until another instruction turns it ON.



- **The RST instruction can be used only with non-retentive variables. With retentive variables, use the RM (Unlatch Retention Coil) instruction.**
- **Real and Integer variables cannot be reset (set to zero) with an RST instruction.**

## 9.2.7 PT (Positive Transition Contact)

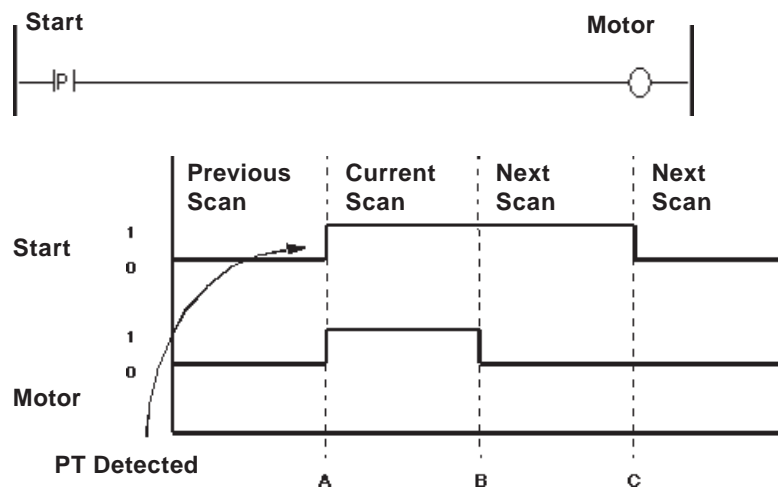
Variable

—|P|—

When the PT instruction is executed, if the variable was OFF during the previous scan but is currently ON, power is allowed to pass for a single scan.

When starting up the program, the state of positive transition contact during the previous scan is considered to have been OFF.

The following diagram is an example of the PT instruction's function.



A: When the Start variable turns ON, the Motor variable turns ON.

B: After one scan (the current scan), the Motor variable turns OFF.

C: Since the rising edge of the variable Start is not detected, the variable Motor remains OFF.

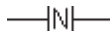


**Be careful when using PT (Rising-type contacts) and NT (Falling-type contacts) instruction operands for indirect addressing of elements in arrays or bit designations via variables.**

**The condition of variables set for operands and used during previous program execution and those variables set for operands are compared and then executed. Therefore, when designated variable values differ, the condition comparison object also differs.**

**9.2.8 NT (Negative Transition Contact)**

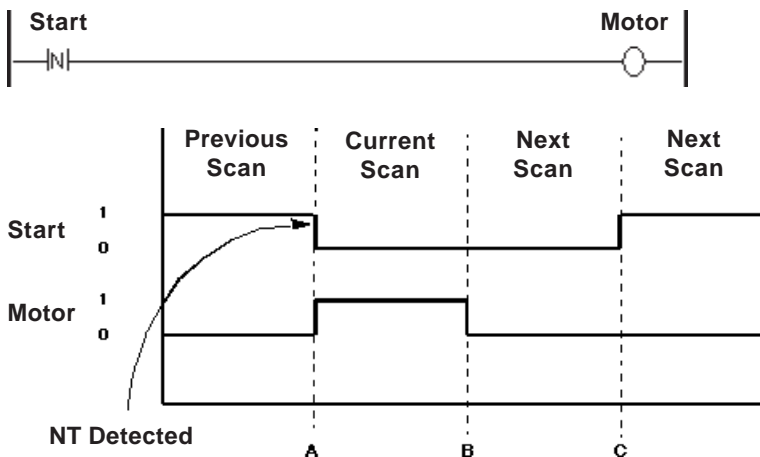
Variable



When the NT instruction is executed, if the variable was OFF during the previous scan but is currently ON, power is allowed to pass for a single scan.

During the first scan, the state of transition during the previous scan is considered to have been OFF. Therefore, the NT instruction does not pass power during the first scan.

The following diagram is an example of the NT instruction's function.



A: When the Start variable turns OFF, the Motor variable turns ON.

B: After one scan, the Motor variable turns OFF.

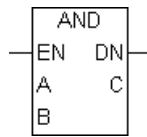
C: Since the falling edge of the variable Start is not detected, the variable Motor remains OFF.



**Be careful when using PT (Rising-type contacts) and NT (Falling-type contacts) instruction operands for indirect addressing of elements in arrays or bit designations via variables.**

**The condition of variables set for operands and used during previous program execution and those variables set for operands are compared and then executed. Therefore, when designated variable values differ, the condition comparison object also differs.**

9.2.9 AND (And)



When the AND instruction is executed, the bit in C turns ON if the corresponding bit in both A and B is ON. Otherwise, the bit in C is turned OFF.

A	Operator	B	C	Integer A
ON	AND	ON	ON	0 1 1 0 ... 1 1 0 0
ON		OFF	OFF	Integer B 1 1 0 0 ... 0 0 0 1
OFF		ON	OFF	Integer C 0 1 0 0 ... 0 0 0 0
OFF		OFF	OFF	

There are three types of AND instructions:

1. When all the variables are not array variables, a simple 32-bit AND operation is performed.
2. When A and C are array variables and B is not an integer array, AND operations are performed for each element of A and B, and the results are stored the corresponding elements of C. Make sure that the size of A and C arrays are the same.
3. When the three variables are arrays of the same size, AND operations of array A and array B are performed. The results are stored in array C.

The AND instruction always passes power.

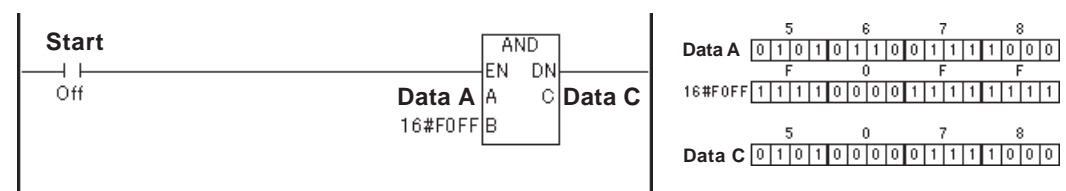
The following table lists the combinations of A, B and C that can be used with an AND instruction.

A	B	C
Integer	Integer	Integer
Integer Array	Integer Array	Integer Array
Integer	Integer Constant	Integer
Integer Array	Integer Constant	Integer Array

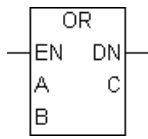
◆ Operation Example

When Start is ON, the 3rd digit of the 4-digit BCD data of Data A is masked to "0", and the result is stored in Data C.

Example) When Data A is "16#5678" (5678 in hexadecimal system), "16#5078" is stored in Data C.



**9.2.10 OR (Or)**



When the OR instruction is executed, the bit in C turns ON if the corresponding bit in A and/or B is ON. Otherwise, the bit in C is turned OFF.

<b>A</b>	<b>Operator</b>	<b>B</b>	<b>C</b>	<b>Integer A</b>	0 1 1 0 ... 1 1 0 0
ON	OR	ON	ON	<b>Integer B</b>	1 1 0 0 ... 0 0 0 1
ON		OFF	ON	<b>Integer C</b>	1 1 1 0 ... 1 1 0 1
OFF		ON	ON		
OFF		OFF	OFF		

There are three types of OR instructions:

1. When both variables A and B are integers, simple 32-bit OR operation is performed.
2. When A and C are array variables and B is not an integer array, logical OR operations are performed for each element of A and B, and the results are stored the corresponding elements of C. Make sure that the size of A and C arrays are the same.
3. When the three variables are arrays of the same size, logical OR operations of array A and array B are performed. The results are stored in array C.

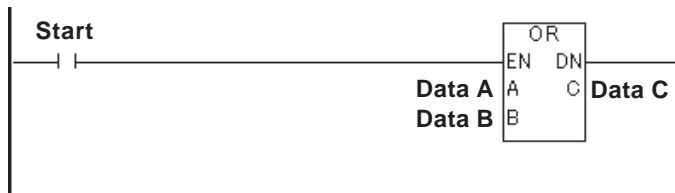
The OR instruction always passes power.

The following table lists the combinations of A, B and C in which OR instructions can be executed.

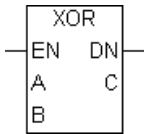
A	B	C
Integer	Integer	Integer
Integer Array	Integer Array	Integer Array
Integer	Integer Constant	Integer
Integer Array	Integer Constant	Integer Array

◆ **Operation Example**

When Start is ON, the result of the logical OR operation of Data A and Data B is stored in Data C.



**9.2.11 XOR (Exclusive OR)**



When the XOR instruction is executed, the bit in C turns ON if the corresponding bit in A or B is ON. Otherwise, the bit in C is turned OFF.

A	Operator	B	C	Integer A	Integer B	Integer C
ON	XOR	ON	OFF	0 1 1 0 ... 1 1 0 0	1 1 0 0 ... 0 0 0 1	1 0 1 0 ... 1 1 0 1
ON		OFF	ON			
OFF		ON	ON			
OFF		OFF	OFF			

There are three types of XOR instructions:

1. When both variables A and B are integers, simple 32-bit exclusive OR operations are performed.
2. When A and C are array variables and B is not an integer array, exclusive OR operations are performed for each element of A and B, and the results are stored the corresponding elements of C. Make sure that the size of A and C arrays are the same.
3. When the three variables are arrays of the same size, exclusive OR operations of array A and array B are performed. The results are stored in array C.

The XOR instruction always passes power.

The following table lists the combinations of A, B and C in which XOR instructions can be executed.

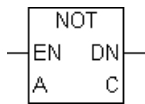
A	B	C
Integer	Integer	Integer
Integer Array	Integer Array	Integer Array
Integer	Integer Constant	Integer
Integer Array	Integer Constant	Integer Array

◆ **Operation Example**

When Start is ON, the result of the exclusive OR operation of Data A and Data B is stored in Data C.



**9.2.12 NOT (Bit Invert)**



When the NOT instruction is executed, the bit in C turns ON if the corresponding bit in A is OFF.

The NOT instruction turns OFF the bit in C if the corresponding bit in A is ON.

<b>A</b>	<b>Operator</b>	<b>C</b>	<b>Integer A</b>	0 1 1 0 ... 1 1 0 0
ON	NOT	OFF	<b>Integer C</b>	1 0 0 1 ... 0 0 1 1
OFF		ON		

There are two types of NOT instruction:

1. When the A variables are integers, simple 32-bit bit conversion is performed.
2. When the A variables are an array, bit conversion is performed for the entire A array. The result is stored in C. Make sure that the size of A and C arrays are the same.

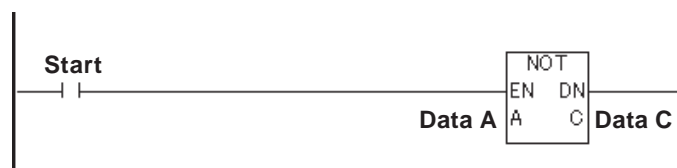
The NOT instruction always passes power.

The following table lists the combinations of A and C in which NOT instructions can be executed.

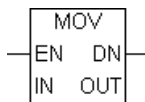
<b>A</b>	<b>C</b>
Integer	Integer
Integer Array	Integer Array

◆ **Operation Example**

When Start is ON, the result of the NOT operation of Data A and Data B is stored in Data C.



**9.2.13 MOV (Transfer)**



When the MOV instruction is executed, IN is copied to OUT.

If IN and OUT are different variable types, the resulting type will be converted to the same type as OUT. To transfer arrays, both IN and OUT must be identical in type and size.

The MOV instruction normally passes power. The following table lists the combinations of IN and OUT in which MOV instructions can be executed.



IN Type	OUT Type
Discrete Array	Discrete array same size as IN
Integer	Variable or Array in Integer or Real
Integer Array	Integer array or variable that is the same size as IN
Integer Constant	Variable or Array in Integer or Real
Real	Variable or Array in Integer or Real
Real Array	Integer array or variable that is the same size as IN
Real Constant	Variable or Array in Integer or Real

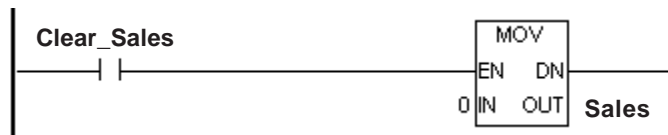


**#Overflow will turn ON if the operation involves a Real-to-Integer data-type conversion, and the value is too large to transfer. In this case, the result will be undefined.**

The following examples illustrate how to use the MOV instruction.

**Example 1: Clear a variable**

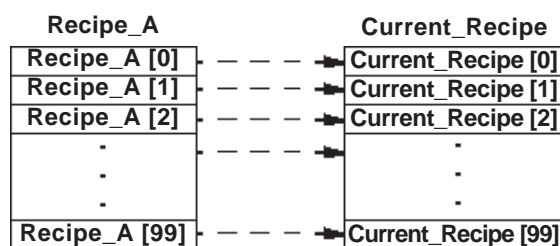
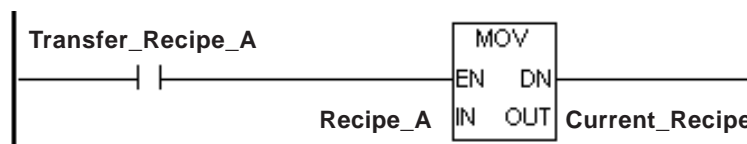
A variable can be cleared with the MOV instruction by transferring a “0” into the variable.



**Example 2: Block-transfer an array**

A block transfer can be performed with the MOV instruction by specifying two arrays of the same type and size.

For example, when transferring Recipe A, which consists of 100 elements, to the Current\_Recipe of the same type and size, simply transfer Recipe A with a MOV instruction.

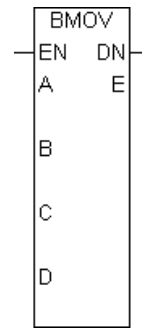


**When designating an entire array, enter only the variable names.**

- E.g.: OK : Recipe\_A
- Not OK : Recipe\_A [\*]
- Not OK : Recipe\_A [100]

**9.2.14 BMOV (Block Transfer)**

- A: Source variable
- B: Start from Array A[B]
- C: To Array E[C]
- D: Amount of data to be transferred
- E: Destination variable



When the BMOV instruction is executed, elements of one array can be copied into elements of another array. Specifically, the D elements are copied from index B in array A to index C in array E.

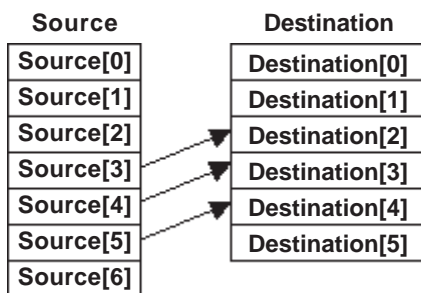
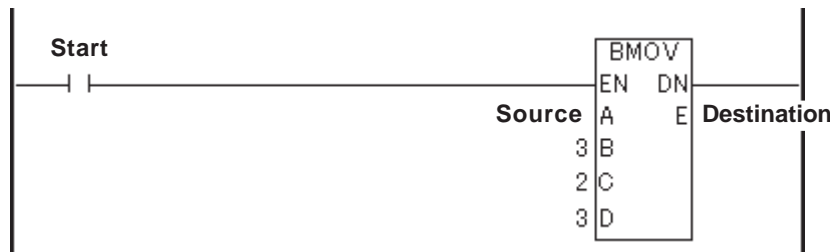
The BMOV instruction is valid for Integer arrays only. When transferring, arrays can be different sizes.

The BMOV instruction always passes power. The following table lists the types of A, B, C, D, and E that can execute BMOV instructions.

A and E	B, C, and D
Integer Array	Integer
	Integer Constant

◆ **Example**

When copying, Source [3], [4], and [5] of the source integer array's 7th element are copied to Destination [2], [3], and [4] of the destination array's 6th element. This data transfer is performed as follows.



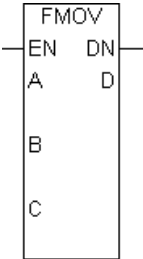
- Source [3] is copied to Destination [2].
- Source [4] is copied to Destination [3].
- Source [5] is copied to Destination [4].

While the program is running, the controller checks whether references to array A and E elements exist in the BMOV instruction. If an invalid array is referred to, a major error occurs and #FaultCode is set to 2.

▼ **Reference** ▲ 8.2.15 #Faultcode

**9.2.15 FMOV (Fill Transfer)**

- A: Source data
- B: Start from Array D[B]
- C: Amount of data to be transferred
- D: Variable name of destination array



When the FMOV instruction is executed, the C elements, starting at index B of Integer array D, are filled with value A.

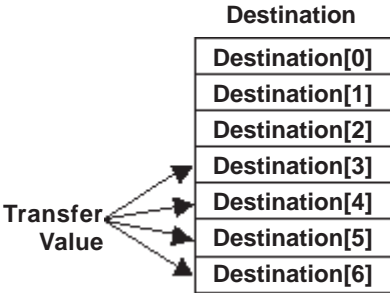
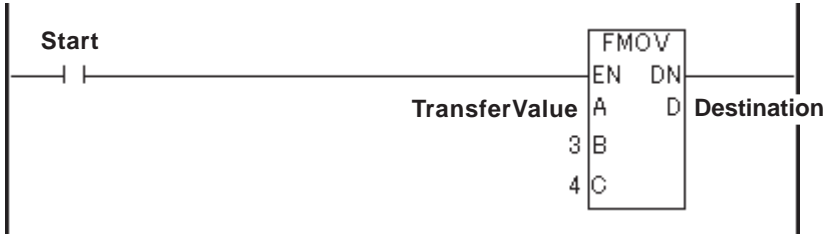
The FMOV instruction is valid for Integer arrays only. The FMOV instruction always passes power.

The following table lists the types of A, B, C and D in which FMOV instructions can be executed.

A, B, and C	D
Integer	Integer Array
Integer Constant	

◆ **Example**

When copying, the values are transferred to Destination [3], [4], [5], and [6] of the destination array's 7th element. The transfer operates as follows.



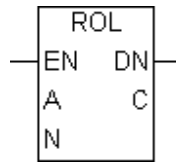
- TransferValue is copied to Destination[3].
- TransferValue is copied to Destination[4].
- TransferValue is copied to Destination[5].
- TransferValue is copied to Destination[6].

While the program is running, the controller checks whether references to array D elements exist in the FMOV instruction. If an invalid array is referred to, a major error will occur and #Faultcode is set to 2.

▼ **Reference** ▲ 8.2.15 #FaultCode

**9.2.16 ROL (Rotate Left)**

A: Variable name to be rotated  
 N: Number of bit positions to shift  
 C: Destination variable



The ROL instruction left-shifts the bits in A by N positions. Bits are rotated from the left end (most significant bit) to the right end (least significant bit). The result is placed in C.

The ROL instruction always passes power.

There are two types of ROL instructions:

1. If both A and C are Integers, a simple 32-bit rotation is performed. N must range from 0 to 31.
2. If both A and C are Integer arrays of the same size, the array is treated as a large Integer.

Bits are shifted from one element to the next, rather than rotating only within each element. N must range from 0 to [(32 x array size) – 1], inclusive.



**Note:** #Overflow is turned ON if N is out of range. The result is undefined.

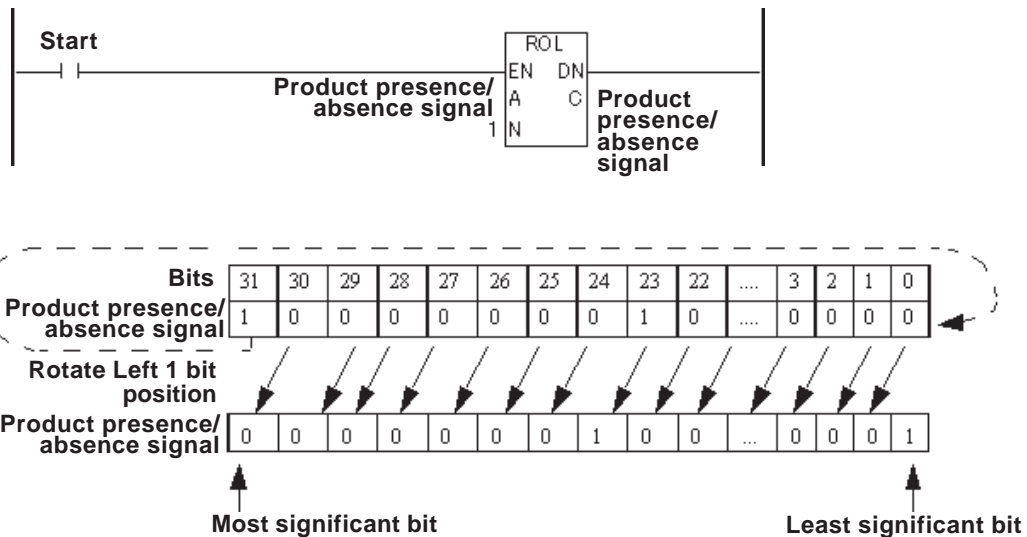
**Reference** 8.2.18 #Overflow

The following table lists the types of A, N and C in which ROL instructions can be executed.

A	N	C
Integer	Integer or Integer Constant	Integer
Integer Array	Integer or Integer Constant	Integer Array is same size as A
Integer Constant	Integer or Integer Constant	Integer

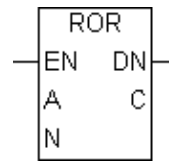
**Example**

The following example describes the operation of a 1-bit rotation using a product presence/absence signal.



**9.2.17 ROR (Rotate Right)**

A: Variable name to be rotated  
 N: Number of bit positions to shift  
 C: Destination variable



The ROR instruction right-shifts the bits in A by N positions. Bits are rotated from the right end (least significant bit) to the left end (most significant bit). The result is placed in C.

The ROR instruction always passes power.

There are two types of ROR instruction.

1. If neither A nor C is an array, a simple 32-bit rotation is performed. N must range from 0 to 31.
2. If both A and C are Integer arrays of the same size, the array is treated as a large Integer. Bits are shifted from one element to the next, rather than rotating only within each element. N must range from 0 to [(32 x array size) – 1], inclusive.



**Note:** #Overflow is turned ON if N is out of range. The result is undefined.

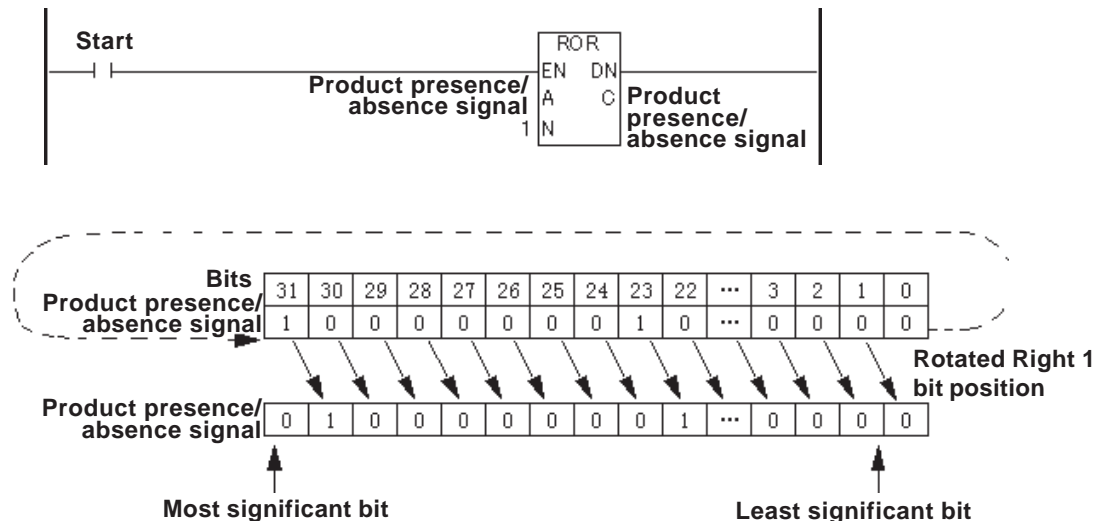
**Reference** 8.2.18 #Overflow

The following table lists the types of A, N and C in which ROR instructions can be executed.

A	N	C
Integer	Integer or Integer Constant	Integer
Integer Array	Integer or Integer Constant	Integer Array is same size as A
Integer Constant	Integer or Integer Constant	Integer

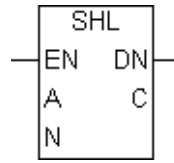
**◆ Example**

The following example describes the operation of 1-bit rotation using the signal of product presence/absence.



**9.2.18 SHL (Shift Left)**

- A: Variable name to be rotated
- N: Number of bit positions to shift
- C: Destination variable



The SHL instruction left-shifts the bits in A by N positions. Bits are dropped from the left end (most significant bit) of the element, and 0 is inserted in the now-vacant bit positions at the right end (least significant bit). The result is placed in C.

There are two types of SHL instruction.

1. If neither A nor C is an array, a simple 32-bit shift is performed. N must range from 0 to 31.
2. If both A and C arrays are the same size, the A array is treated as a large Integer. Bits are shifted from one element to the next, rather than the most significant bit being dropped from the left end of each element. Only the most significant bit of the highest-numbered element within the array is dropped. N must range from 0 to [(32 x array size) – 1], inclusive.



**Note:** #Overflow is turned ON if N is out of range. The result is undefined.

**Reference** 8.2.18 #Overflow

The SHL instruction always passes power.

The following table lists the types of A, N and C in which SHL instructions can be executed.

A	N	C
Integer	Integer or Integer Constant	Integer
Integer Array	Integer or Integer Constant	Integer Array is same size as A
Integer Constant	Integer or Integer Constant	Integer

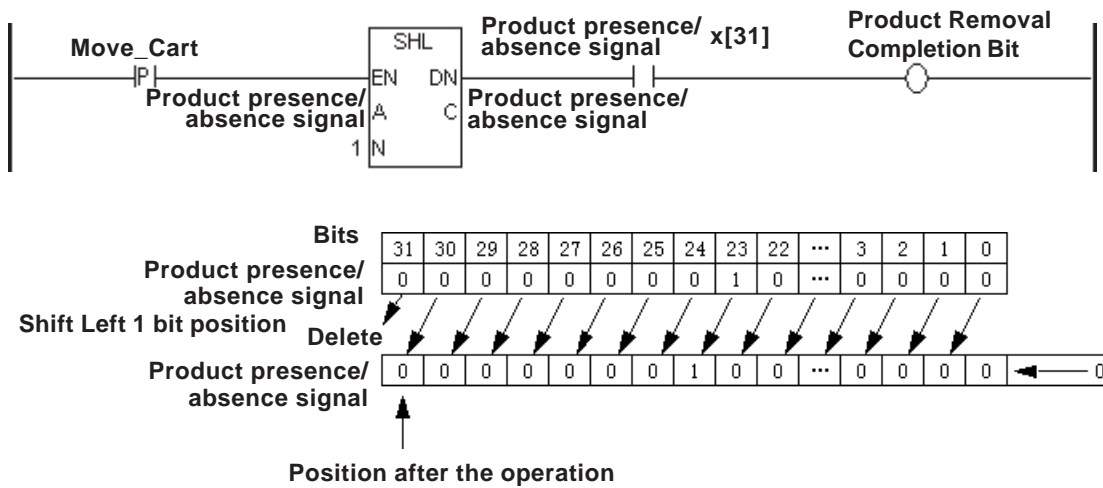
◆ **Example**

The following diagram is an example of a one-bit left shift, used to track the position of a bit.

Each bit in the product presence/absence signal represents the actual position of the product.

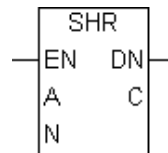
When "Move\_Cart" is turned ON, bit is shifted left to the next position.

When the bit reaches the final bit position in the variable (31), the Product Removal Completion Bit is turned ON, indicating that the operation is completed.



**9.2.19 SHR (Shift Right)**

- A: Variable name to be rotated
- N: Number of bit positions to shift
- C: Destination variable



The SHR instruction right-shifts the bits in A by N positions. Bits are dropped from the right end (least significant bit) of the element, and 0 is inserted in the now-vacant bit positions at the left end (most significant bit). The result is placed in C.

There are two types of SHR instructions.

1. If neither A nor C is an array, a simple 32-bit shift is performed. N must range from 0 to 31.
2. If both A and C arrays are the same size, the A array is treated as a large Integer. Bits are shifted from one element to the next, rather than the least significant bit being dropped from the right end of each element. Only the least significant bit of the lowest-numbered element within the array is dropped. N must range from 0 to [(32 x array size) – 1], inclusive.



**#Overflow is turned ON if N is out of range. The result is undefined.**

**Reference** 8.2.18 #Overflow

The SHR instruction always passes power. The following table lists the types of A, N and C in which SHR instructions can be executed.

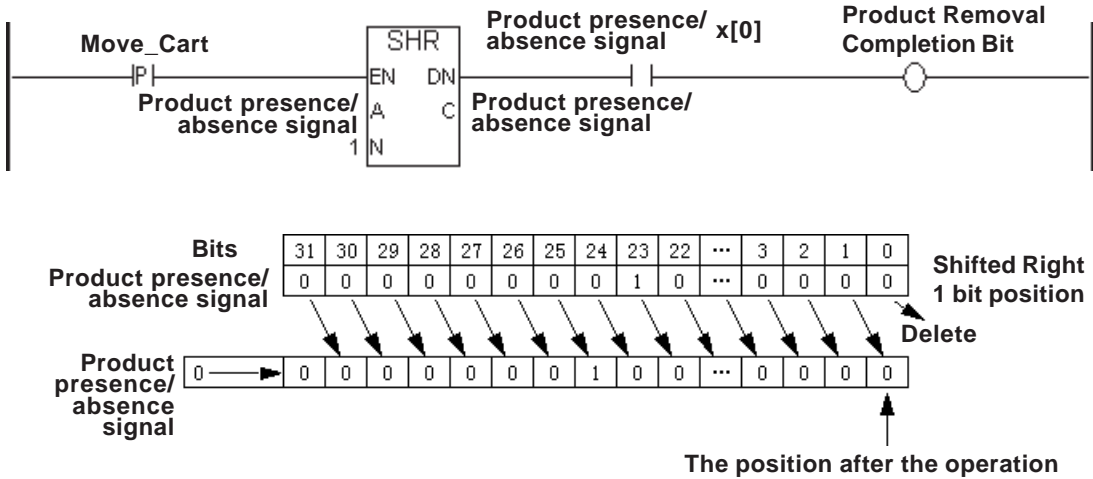
A	N	C
Integer	Integer or Integer Constant	Integer
Integer Array	Integer or Integer Constant	Integer Array is same size as A
Integer Constant	Integer or Integer Constant	Integer

# Chapter 9 – Instructions

## ◆ Example

### • When Using Bits

The following diagram is an example of a one-bit right shift, used to track the position of a bit. Each bit in the product presence/absence signal represents the actual position of the product. When "Move\_Cart" is turned ON, bit is shifted right to the next position. When the bit reaches the final bit position in the variable (0), the Product Removal Completion Bit is turned ON, indicating that the operation is completed.

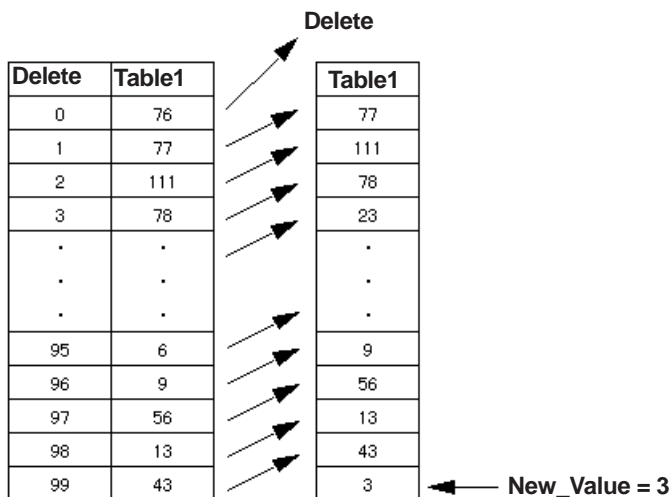
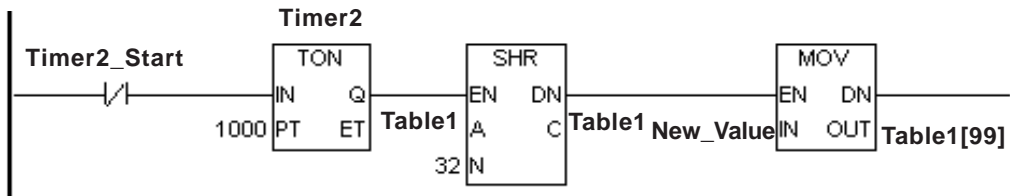


### • When Using Arrays

The following diagram is an example an SHR instruction being used to transfer values of each element in an Integer array.

A 32-bit shift rotates the entire 32-bit Integer.

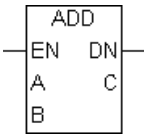
Every second, the "Table1" Integer array's values are moved up one position towards 0 and a new value is placed at the end of the elements "Table1[99]" in the "Table1" Integer array.





**9.2.20 ADD (Add)**

- A: Data
- B: Data
- C: Destination Variable



When the ADD instruction is executed, A and B are added, and the result is placed in C.

If both A and B are Integers or Integer constants, the ADD instruction performs an Integer addition. Otherwise, the instruction performs a floating-point instruction, which may reduce the processing speed.

The ADD instruction always passes power. The following table lists the combinations of A, B and C in which ADD instructions can be executed.

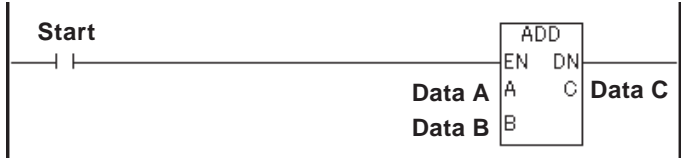
A	B	C
Integer	Integer	Integer or Real
Integer Constant	Integer Constant	Integer or Real
Real	Real	Integer or Real
Real Constant	Real Constant	Integer or Real



- Note:**
  - If the result C exceeds the range expressed with the variable data type in C, #Overflow turns ON and the result of ADD is undefined.
    - ▼ **Reference** 8.2.18 #Overflow
  - If either A or B are Real, both are converted to Real prior to the addition. However, if C is an Integer, the number is truncated after the decimal point, since the result is placed in C.

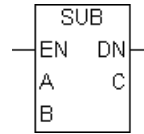
◆ **Example**

When Start is turned ON, Data A and Data B are added and the result of the operation is stored in Data C.



**9.2.21 SUB (Subtract)**

A: Data  
 B: Data  
 C: Destination Variable



When the SUB instruction is executed, B is subtracted from A, and the difference is placed in C.

If both A and B are Integers or Integer constants, the SUB instruction performs an Integer subtraction. Otherwise, the instruction performs a floating-point instruction, which may reduce the processing speed.

The SUB instruction always passes power. The following table lists the types of A, B and C in which SUB instructions can be executed.

A	B	C
Integer	Integer	Integer or Real
Integer Constant	Integer Constant	Integer or Real
Real	Real	Integer or Real
Real Constant	Real Constant	Integer or Real



**Note:**

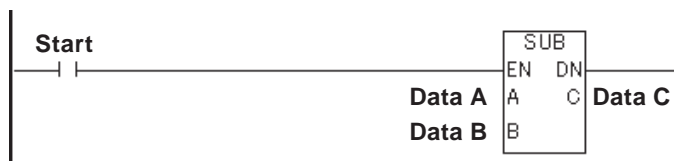
- If the result C exceeds the range expressed with the variable data type in C, #Overflow turns ON and the result of SUB is undefined.

**Reference** 8.2.18 #Overflow

- If either A or B are Real, both are converted to Real prior to the subtraction. However, if C is an Integer, the number is truncated after the decimal point, since the result is placed in C.

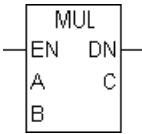
◆ **Example**

When Start is turned ON, Data B is subtracted from Data A and the result of the operation is stored in Data C.



**9.2.22 MUL (Multiply)**

- A: Data
- B: Data
- C: Destination variable



When the MUL instruction is executed, A is multiplied by B, and the result is placed in C. If both A and B are Integers or Integer constants, the MUL instruction performs an Integer multiplication.

Otherwise, the instruction performs a floating-point instruction, which may reduce the processing speed.

The MUL instruction always passes power. The following table lists the combinations of A, B and C in which MUL instructions can be executed.

A	B	C
Integer	Integer	Integer or Real
Integer Constant	Integer Constant	Integer or Real
Real	Real	Integer or Real
Real Constant	Real Constant	Integer or Real



**Note:**

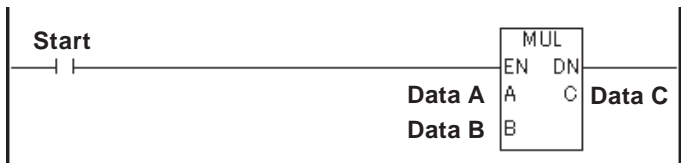
- If the result C exceeds the range expressed by the variable data type in C, #Overflow turns ON and the result of MUL is undefined.

**Reference** 8.2.18 #Overflow

- If either A or B are Real, both are converted to Real prior to the multiplication. However, if C is an Integer, the number is truncated after the decimal point, since the result is placed in C.

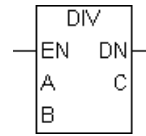
◆ **Example**

When Start is turned ON, Data A is multiplied by Data B, and then the result of the operation is stored in Data C.



**9.2.23 DIV (Divide)**

- A: Data
- B: Data
- C: Destination variable



When the DIV instruction is executed, A is divided by B, and the quotient is placed in C.

If both A and B are Integers or Integer constants, the DIV instruction performs an Integer multiplication. Otherwise, the instruction performs a floating-point instruction, which may reduce the processing speed.

The DIV instruction always passes power.

The following table lists the combinations of A, B and C in which DIV instructions can be executed.

A	B	C
Integer	Integer	Integer or Real
Integer Constant	Integer Constant	Integer or Real
Real	Real	Integer or Real
Real Constant	Real Constant	Integer or Real



**Note:**

- If B is zero or if the result C exceeds the range expressed by the variable data type in C, #Overflow turns ON and the result of DIV is undefined.

**Reference** 8.2.18 #Overflow

- If either A or B are Real, both are converted to Real prior to the division. However, if C is an Integer, the number is truncated after the decimal point, since the result is placed in C.

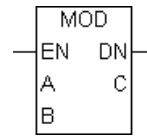
◆ **Example**

When Start is turned ON, Data A is divided by Data B and the result of the operation is stored in Data C.



**9.2.24 MOD (Modulus)**

- A: Data
- B: Data
- C: Destination variable



When the MOD instruction is executed, A is divided by B, and the remainder is placed in C. The MOD instruction performs only Integer or Integer Constant operations.

The MOD instruction always passes power.

The following table lists the combinations of A, B and C in which MOD instructions can be executed.

A	B	C
Integer Constant	Integer	Integer
Integer	Integer Constant	Integer



**Note:** #Overflow is turned ON when divided by zero, and the result C is undefined.

**Reference** 8.2.18 #Overflow

◆ **Example**

When Start is turned ON, Data A is divided by Data B and the remainder is stored in Data C.

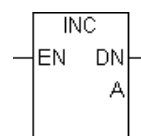


The following example is an Integer (27) divided by 5, and the result (2) is placed in C.

$$\begin{array}{r}
 \text{A}=27 \quad 5 \overline{)27} \\
 \text{B}=5 \quad \underline{25} \\
 \quad \quad 2 \leftarrow \text{C}=2
 \end{array}$$

**9.2.25 INC (Increment)**

- A: Data



When the INC instruction is executed, one (1) is added to A, and the result is then placed in A.

The INC instruction always passes power.



**Note:** #Overflow is set if A increments from 0x7FFFFFFF to 0x80000000.

**Reference** 8.2.18 #Overflow

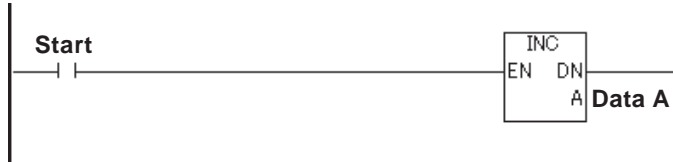
## Chapter 9 – Instructions

The following table lists the combinations of A in which INC instructions can be executed.

<b>A</b>
Integer

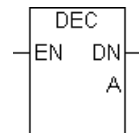
### ◆ Example

When Start is turned ON, "1" is added to Data A.



### 9.2.26 DEC (Decrement)

A: Data



When the DEC instruction is executed, one (1) is subtracted from A, and the result is then placed in A.

The DEC instruction always passes power.

The following table lists the combinations of A in which DEC instructions can be executed.

<b>A</b>
Integer



**Note:** #Overflow is set if A decrements from 0x80000000 to 0x7FFFFFFF.

▼ **Reference** ▲ 8.2.18 #Overflow

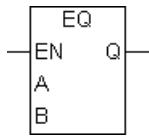
### ◆ Example

When Start is turned ON, "1" is subtracted from Data A.



**9.2.27 EQ (Compare: = )**

A: Data  
B: Data



The EQ instruction passes power if A is equal to B.  
The following table lists the combinations of A and B in which EQ instructions can be executed.

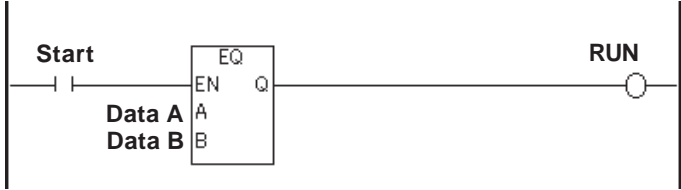
A	B
Integer	Integer
Integer Constant	Integer Constant
Real	Real
Real Constant	Real Constant



**Note:** Real values need to be compared very carefully. For example, a calculation might result in 1.9999999999, which is not equal to 2.0000000000.

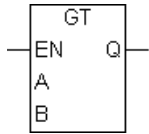
◆ **Example**

Run mode is triggered when the values of Data A and Data B are equal after Start is turned ON.



**9.2.28 GT (Compare: > )**

A: Data  
B: Data



The GT instruction passes power if A is greater than B.  
The following table lists the combinations of A and B in which GT instructions can be executed.

A	B
Integer	Integer
Integer Constant	Integer Constant
Real	Real
Real Constant	Real Constant

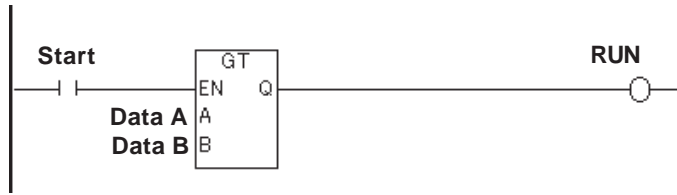


**Note:** Real values need to be compared very carefully. For example, a calculation might result in 2.000000000001, which is greater than 2.

## Chapter 9 – Instructions

### ◆ Example

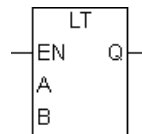
Run mode is triggered when the value of Data A is greater than that of Data B after Start is turned ON.



### 9.2.29 LT (Compare: < )

A: Data

B: Data



The LT instruction passes power if A is less than B.

The following table lists the combinations of A and B in which LT instructions can be executed.

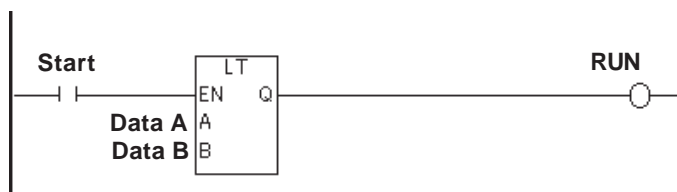
A	B
Integer	Integer
Integer Constant	Integer Constant
Real	Real
Real Constant	Real Constant



**Note:** Real values need to be compared very carefully. For example, a calculation might result in 1.9999999999, which is less than 2.

### ◆ Example

Run mode is triggered when the value of Data A is smaller than that of Data B after Start is turned ON.

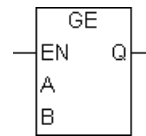




**9.2.30 GE (Compare: >= )**

A: Data

B: Data



The GE instruction passes power if A is greater than or equal to B.

The following table lists the combinations of A and B in which GE instructions can be executed.

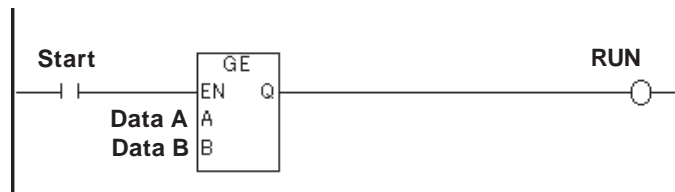
A	B
Integer	Integer
Integer Constant	Integer Constant
Real	Real
Real Constant	Real Constant



**Note:** Real values need to be compared very carefully. For example, a calculation might result in 1.9999999999, which is not greater than or equal to 2.

◆ **Example**

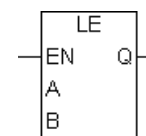
Run mode is triggered when the value of Data A is equal to or greater than that of Data B after Start is turned ON.



**9.2.31 LE (Compare: <= )**

A: Data

B: Data



The LE instruction passes power if A is less than or equal to B.

The following table lists the combinations of A and B in which LE instructions can be executed.

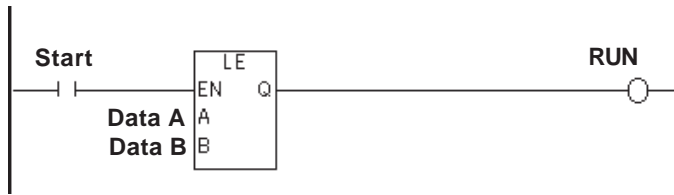
A	B
Integer	Integer
Integer Constant	Integer Constant
Real	Real
Real Constant	Real Constant



**Note:** Real values need to be compared very carefully. For example, a calculation might result in 2.00000000001, which is not less than or equal to 2.

◆ Example

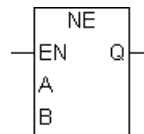
Run mode is triggered when the value of Data A is equal to or smaller than that of Data B after Start is turned ON.



**9.2.32 NE (Compare: <> )**

A: Data

B: Data



The NE instruction passes power if A is not equal to B.

The following table lists the combinations of A and B in which NE instructions can be executed.

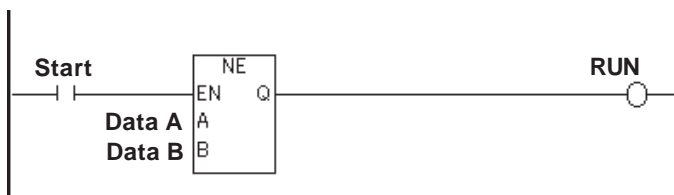
A	B
Integer	Integer
Integer Constant	Integer Constant
Real	Real
Real Constant	Real Constant



**Note:** Real values need to be compared very carefully. For example, a calculation might result in 1.9999999999, which is not equal to 2.

◆ Example

After Start is turned ON, Run mode is triggered when the values of Data A and Data B are not equal.



When power is passed to the timer starting bit (IN), the TON instruction starts, and:

- Variable.ET (the elapsed time) begins to increment in milliseconds.
- Variable.TI (the timing bit) turns ON.
- Variable.Q (the timer output bit) turns OFF.

When the elapsed time (Variable.ET) increments and equals the preset time (Variable.PT):

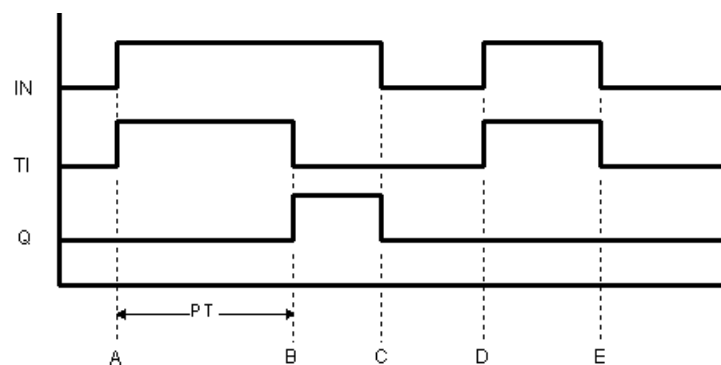
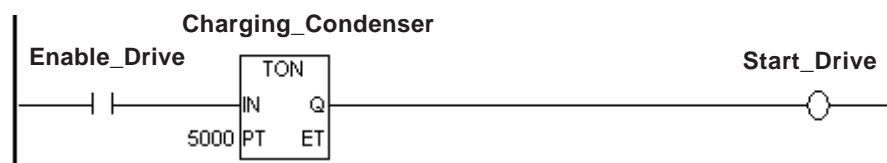
- Variable.ET (the elapsed time) holds the current value.
- Variable.TI (the timing bit) turns OFF.
- Variable.Q (the timer output bit) turns ON, and the instruction passes power.

When the timer starting bit (IN) stops passing power to start the TON instruction:

- Variable.ET (the elapsed time) is reset to zero.
- Variable.TI (the timing bit) turns OFF.
- Variable.Q (the timer output bit) turns OFF.

### ◆ Example

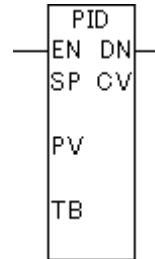
In the following example, the drive will be started 5 seconds after "Enable\_Drive" is turned ON.



- A: When power is applied to the timer input bit (IN), the timing bit (TI) turns ON, the timer begins timing, and the elapsed time (ET) increments. The timer output bit (Q) remains OFF.
- B: The elapsed time (ET) equals the preset time (PT), the timer output bit (Q) turns ON, and the elapsed time (ET) stays fixed at the preset time. The timing bit (TI) turns OFF.
- C: The timer input bit (IN) turns OFF, the timer output bit (Q) turns OFF, and the elapsed time (ET) is reset to 0.
- D: The timer input bit (IN) turns ON, and the timing bit (TI) turns ON. The timer begins timing, and the elapsed time (ET) increments.

**9.2.33 PID (PID Calculation)**

SP: Setpoint  
 PV: Process Variable  
 TB: Tieback  
 CV: Control Variable



The PID (Proportional Integral Derivative) instruction compares a measured value (Process Variable), from the analog input or temperature input, with a preset value (Setpoint). The PID then adjusts the Control Variable to eliminate the difference between the Process Variable and the Setpoint.

When performing the PID control, the proportional (P), the integral (I), and the derivative (D) controls can be combined freely. By setting each parameter (described later in this section), these controls can be executed.

The control value calculated by the PID control can be expressed in the following equation.

$$CV = KC(E + \text{Reset} \int_0^t (E)dt + \text{Rate} \frac{d(E)}{dt})$$

- KC : Proportional Coefficient\*1
- E : Error Signal (SP-PV or PV-SP)
- Reset: Integral Time\*1
- Rate : Derivative Time\*1

By adjusting the sampling period in the Tune tab which is described later, the effect of noise on the error signal can be reduced. The filtered error signal can be expressed in the following equation.

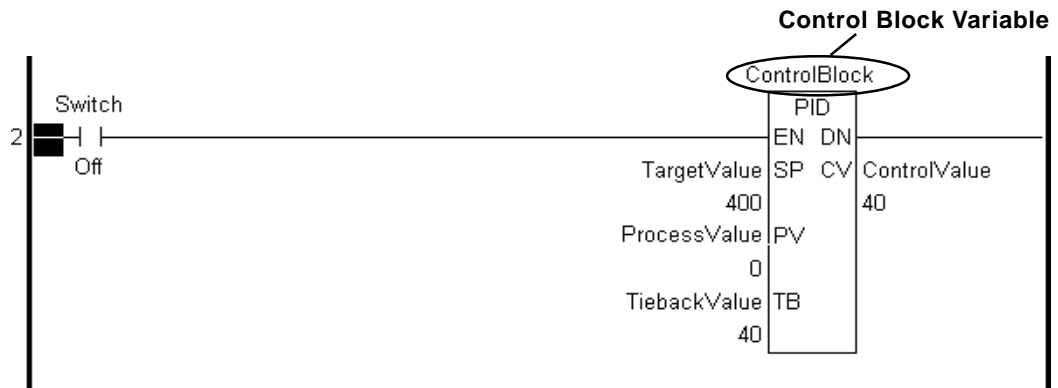
$$EF_n = EF_{n-1} + \frac{T_{Loop}}{T_{Filter}} (E_n - EF_{n-1})$$

- EF : Filtered Error Signal
- Tloop : Data Refresh Period (Loop update time)
- TFilter : Sampling Time (DFTC)
- E : Error Signal (SP-PV or PV-SP)

\*1 This is set in the “Tuning” tab, explained on the following page. This is not a control block variable value.

■ Overview

When the PID instruction passes power, it adjusts the PID output (Automatic Mode). If the PID instruction is not passing power, a constant control amount is output (Manual mode). In Manual mode, the Control Variable is set to the Tieback value.



When using the PID instruction in a logic program, map variables to the control block, SP, PV, TB, and CV variables.

◆ Parameter and Variable Type

Parameter	Description	Variable Type
SP	Setpoint	Integer, Integer Constant, Integer Array
PV	Process variable	Integer, Integer Array
TB	Tieback When the instruction doesn't receive power, value set in this is output.	Integer, Integer Constant, Integer Array
CV	Controlled variable	Integer, Integer Array

### ◆ Control Block Variable

When a variable is mapped to the PID instruction, an array with seven elements (see following table) is mapped to the variable. Element [0] represents the current status, and Elements [1] to [6] are used for the PID control to make fine-tuning adjustments.

Element Number	Description	
0	Bit 0	Mode Switch Flag
	Bit 1	PID instruction process completion flag
	Bit 2	PID deadband flag
	Bit 3	Control variable exceeds upper limit
	Bit 4	Control variable exceeds lower limit
	Bit 5	Exceeded the number of integration process times
1	Proportional coefficient	
2	Integration times per minute	
3	Derivative time per cycle	
4	PID deadband	
5	Offset	
6	Sampling time	



**Note:**

The variable type of a control block variable will be retentive.



**Important**

*The values in the control block variable for the proportional coefficient, the number of integral times per minute, and the derivative time per cycle are 1000 times the values of proportional coefficient, integral times, and derivative time set in the Tune tab.*

### ■ Control Block Variable Element [0] Status

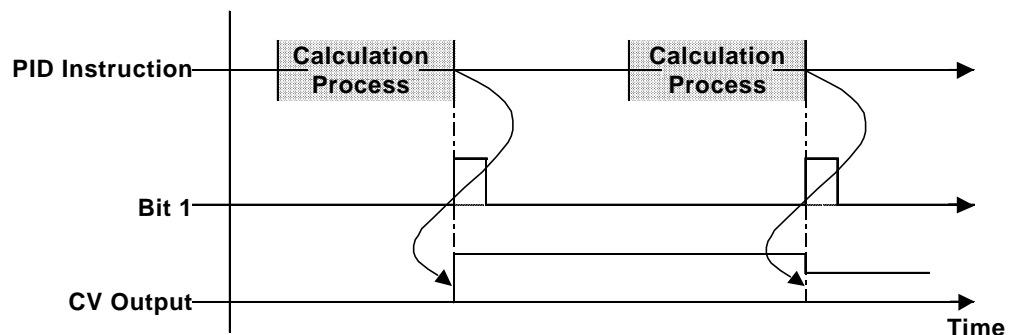
#### ◆ Mode Switch Flag (Bit 0)

When the PID instruction in a logic program passes power, bit 0 turns ON.

Bit 0	Mode
ON	Automatic Mode (PID Calculation)
OFF	Manual Mode

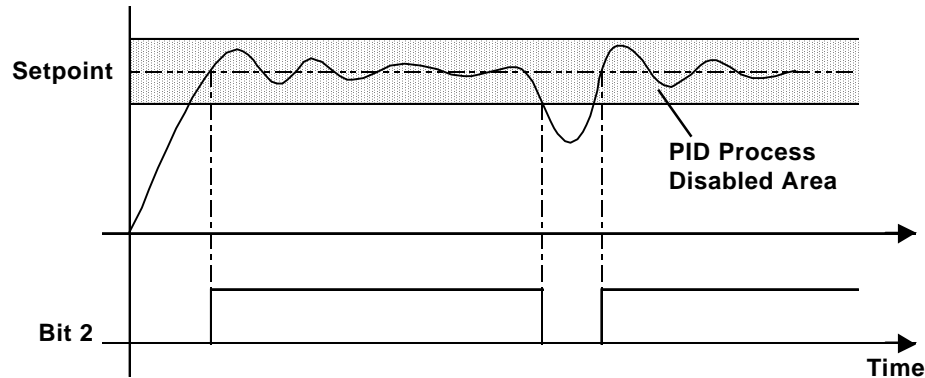
#### ◆ PID Instruction Process Completion Flag (Bit 1)

When the calculation process is finished and the CV is output, bit 1 turns ON. Bit 1 stays ON during one scan.



◆ **PID Deadband Flag (Bit 2)**

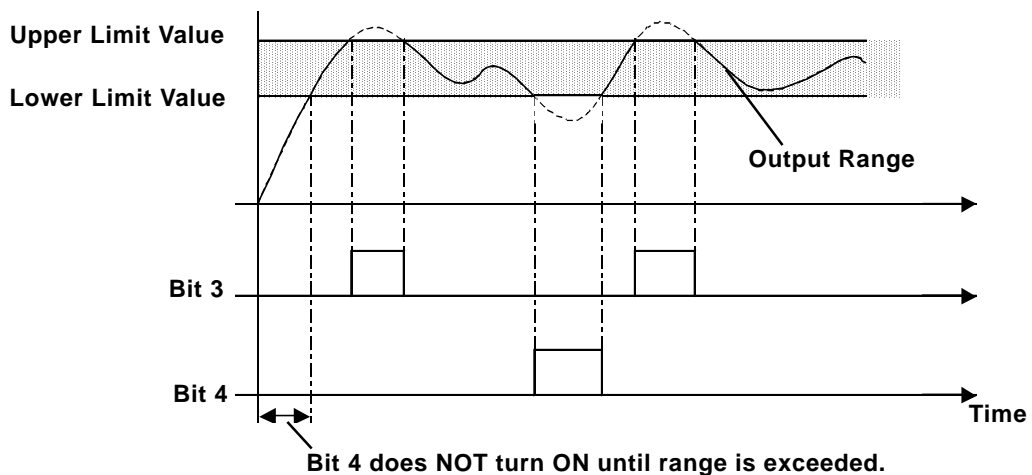
When the control variable is within the range specified in the Tune tab of the [PID] dialog box, or in Element [4] in the control block variable, bit 2 turns ON when the process variable reaches the setpoint. Bit 2 turns OFF when the process variable is outside the range.



**Note:** For details about the Tune tab in the [PID] dialog box, see the following ■ *Fine-Tuning Adjustments and Monitoring of PID Control* section.

◆ **Control Variable Exceeds the Upper Limit (Bit 3) or Lower Limit (Bit 4)**

When there is an output at the upper limit specified in the [Tune] tab of the [PID] dialog box, bit 3 turns ON. When there is an output at the lower limit, bit 4 turns ON. Even if a status bit is turned ON, PID calculation is performed and either the upper limit value or lower limit value will be output.



**Note:** For details about the Setup tab in the [PID] dialog box, see the following ■ *Fine-Tuning Adjustments and Monitoring of PID Control* section.

◆ **Exceeding the Number of Integration Process Times (Bit 5)**

When processing is performed for an integration frequency that is outside the range assigned in the Tune tab of the [PID] dialog box, bit 5 turns ON. Even if this status bit is turned ON, PID calculation is performed and the value is output at the upper limit.



**Note:** For details about the Setup tab in the [PID] dialog box, see the following ■ *Fine-Tuning Adjustments and Monitoring of PID Control* section.

■ **Control Block Variable Elements [1]–[6] Status**

Elements [1]-[6] perform fine-tuning adjustments of PID control.

▼ **Reference** ▲ For details, see the following ■ *Fine-Tuning Adjustments and Monitoring of PID Control* section.

■ **Fine-Tuning Adjustments and Monitoring of PID Control**

Clicking the instruction after setting up special variables and control block variables to the PID instruction displays the following [PID] dialog box. Fine-tuning adjustments and monitoring of the PID control settings are available in this dialog box.



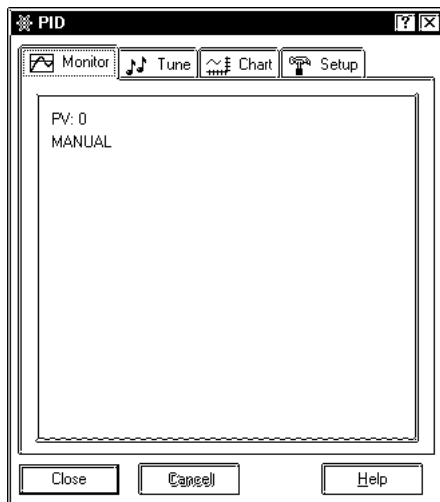
**Note:**

There is no tuning feature to automatically adjust each parameter.

◆ **Monitor**

While in Monitoring mode, use the Monitor feature to monitor the PID instruction execution result.

▼ **Reference** ▲ *Pro-Control Editor Operation Manual Chapter 4 Online Editing*



**Type of Chart Lines**

The following table lists the types of lines and colors of each item monitored.

Item	Types of Lines/Colors	
Setpoint	Black dotted line	— · — · — · — · —
Process Variable	Black solid line	—————
Control Variable	Blue solid line	—————



**Note:**

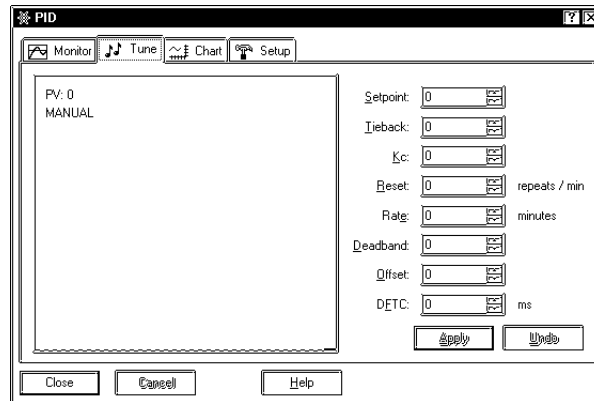
Types and colors of graph lines cannot be changed.



◆ **Tune**

Each value can be adjusted while monitoring. The values set here reflect the special variables or control block variable elements [1] to [6].

**Reference** *Pro-Control Editor Operation Manual Chapter 4 Online Editing*



**Setpoint**

Sets the target value.

**Tieback**

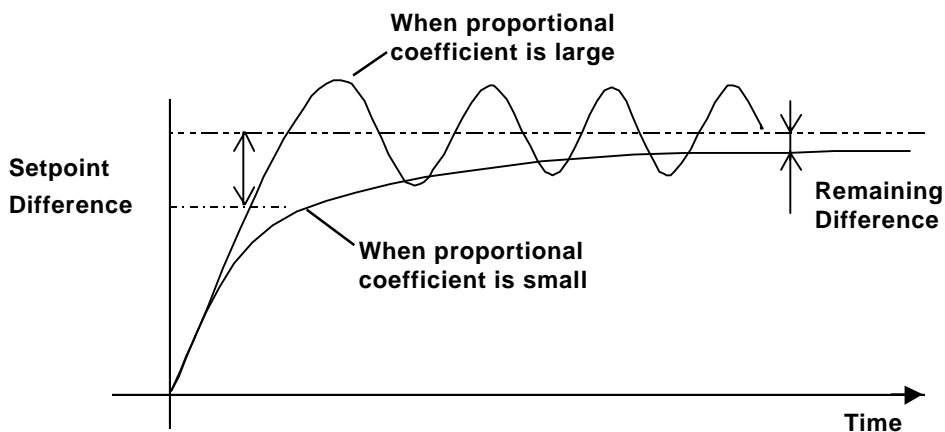
When the PID instruction in the logic program does NOT pass power, the value set here will be output.

**Proportional Coefficient [KC]**

The Control Variable's output is based on the difference between the Process Variable and the Setpoint.

When the proportional coefficient is decreased, the control amount for bringing the Process Variable closer to the target value decreases, and overshoot is prevented. However, this may increase the remaining difference.

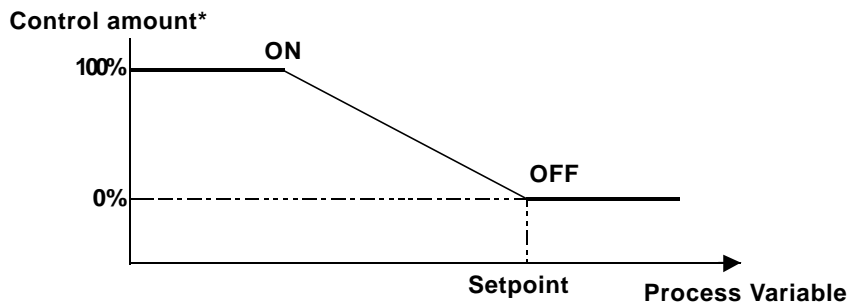
When the proportional coefficient is increased, the control amount for bringing the current value closer to the target value increases, and the length of time to reach the target value will shorten. However, this may cause hunting.



## Chapter 9 – Instructions



**Note:** When using proportional control, when the Process Variable is smaller than the Setpoint and the control amount reaches its maximum limit at 100%, if the Setpoint and the Process Variable match (no difference), the control amount becomes 0%.



\* Control amount: output per unit time

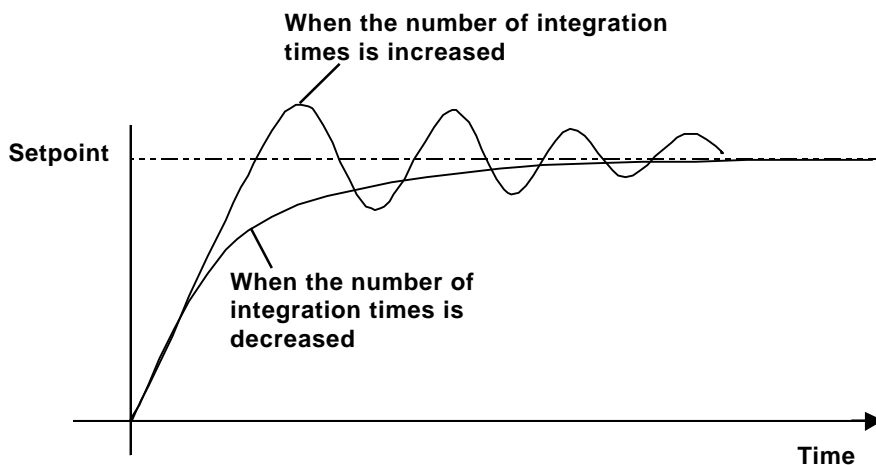
### Number of Integration Times [Reset]

The control amount alone can never completely eliminate the difference, since the control amount (control output) becomes too small when it gets close to the Setpoint. Using integral control, that remaining difference can be eliminated.

This control method makes adjustments based on the accumulated difference, over time, between the Process Variable and the Setpoint. If it reaches a certain level, it affects the output to reduce the difference.

When the number of integration times is increased, the control amount to reduce the difference increases. The length of time to reach the Setpoint will shorten. However, this may cause overshoot and hunting.

When the number of integration times is decreased, the control amount to reduce the difference decreases. Overshoot and hunting are eliminated. However, the length of time to reach the Setpoint will be greater.



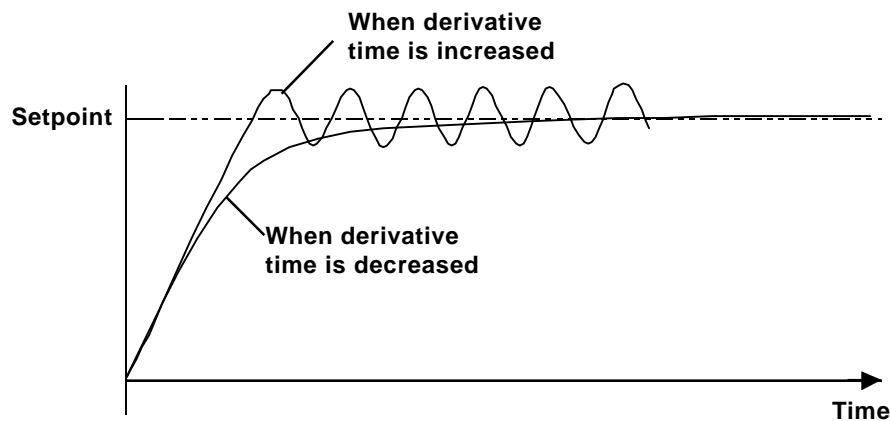
### Derivative Time [Rate]

Proportional control or integral control that requires a certain time (time constant) cannot respond quickly to a disturbance, and cannot return to the target value quickly.

Derivative control monitors the difference against the disturbance, and when a difference is large compared to the previous difference, a large amount of control is given to provide a quick response.

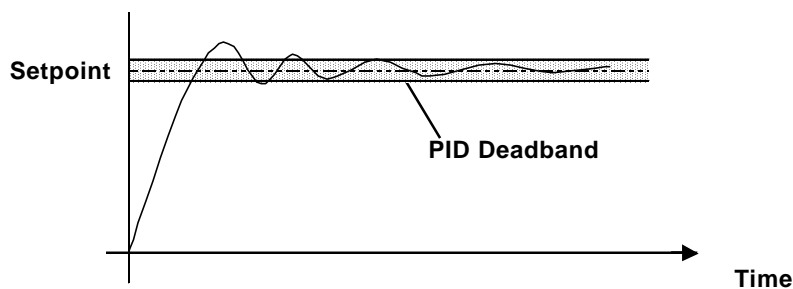
When the derivative time is increased, recovery time from the disturbance is shortened. However, this may cause overshoot and short cycle hunting.

When the derivative time is decreased, overshoot and hunting are eliminated. However, the recovery time from the disturbance will be greater.



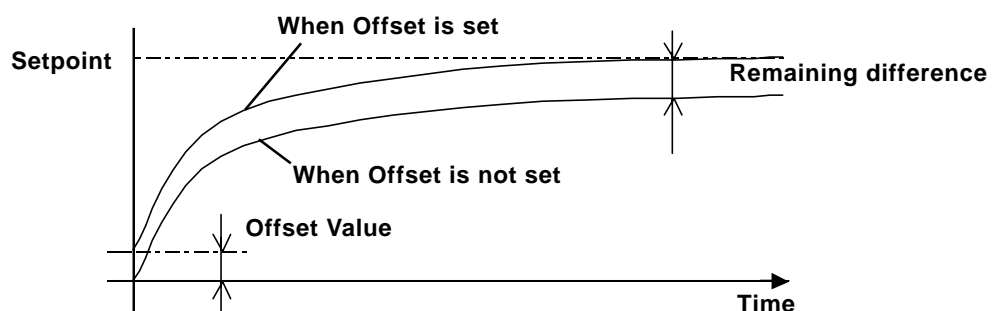
### Deadband

PID control is not performed in the deadband, and the minimum control variable value is output, which provides smooth control without hunting.



### Offset

Set offset value. Offset can reduce the remaining difference created by the proportional control.



## Chapter 9 – Instructions

### Sampling Time [DFTC]

Provides noise reduction of the connected device's measured data acquired by the Loop Update Time. Calculates a running average of the previous filtering result and the currently acquired data.

Setting the sampling time allows for the measured data to contain unexpected measurements. If the previously measured data is calculated as an average, the effect of unexpected measurements on the output value will be minor.

Sampling Time should be set to a larger value than the Loop Update Time. Also, setting the sampling value to "0" will disable the filter.

For loop update time information, refer to "Control" in the "Setup" tab.

### Reflect Tune tab setting to Control Block Variable

Each setting in the Tune tab is reflected in the parameter variables (SP and TB) and the elements [1] to [6] of the control block variable.

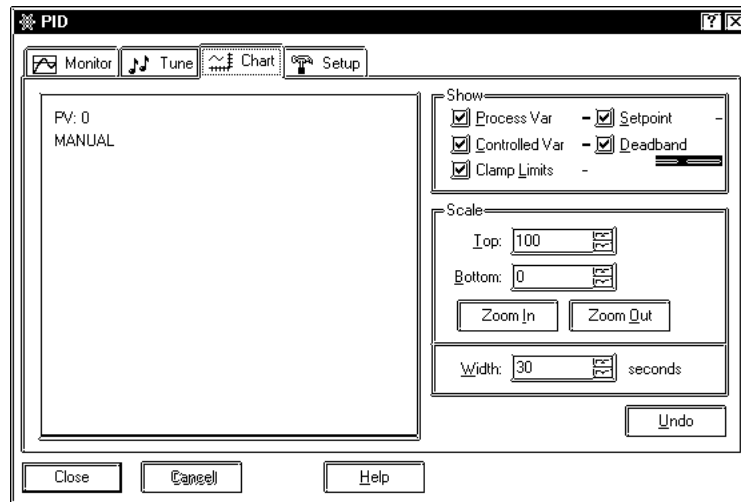
The following tables compare the Tune tab and Parameter variables, and the Tune tab and control block variable. 1000 times the values in the proportional coefficient, integral times, and derivative period are written in the control block variable.

Tune Tab	Scale Factor	➔	Parameter Variables	
Setpoint	x 1		SP	Variable
Tieback	x 1		TB	Variable

Tune Tab	Scale Factor	➔	Control Block Variable	
Proportional Coefficient	x 1000		Proportional Coefficient	Control Block Variable [1]
Integral number of times	x 1000		Integral number of times per minute	Control Block Variable [2]
Derivative period	x 1000		Derivative number of times per operation	Control Block Variable [3]
Deadband	x 1		PID deadband	Control Block Variable [4]
Offset	x 1		Offset	Control Block Variable [5]
Sampling time	x 1		Sampling time	Control Block Variable [6]

◆ **Chart**

Process Variable (PV), Setpoint (SP), Control Variable (CV), Deadband, and Clamp Limits can be monitored. The monitoring setup is available in the Chart tab.



**Show**

Chart types and lines of each item monitored are listed in the following table.

Item	Types of Lines/Colors	
Setpoint	Black dotted line	— — — — —
Process Variable	Black solid line	—————
Controlled Variable	Blue solid line	—————
Clamp Limits	Red dotted line	- - - - -
Deadband	Gray zone	



**Note:** Types of lines/colors cannot be changed.

**Scale**

Top: Set the upper limit of the chart

Bottom: Set the lower limit of the chart

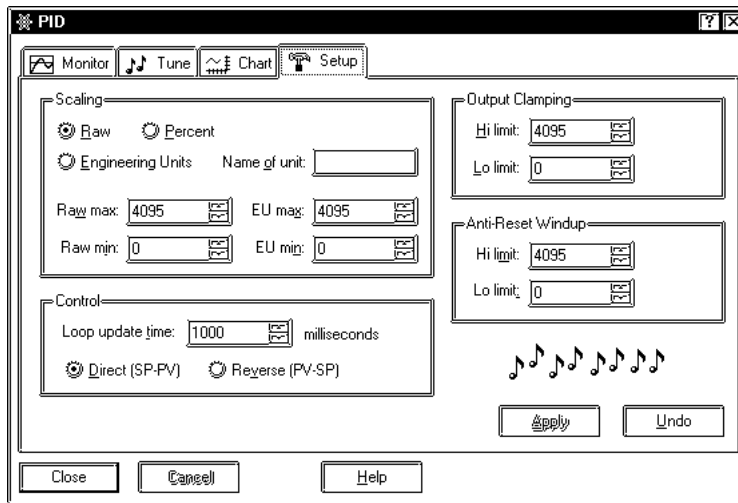
Width: Set the width of the chart in seconds. Sampling time can be changed in the Preference area of the Monitoring tab by clicking Option in the Editor's File menu.



**Note:** Previous data cannot be monitored.

### ◆ Setup

You can preset the range (upper and lower limits) set to all parameters during the programming mode.



**Note:** These settings are not available during the monitoring mode.

### Scaling

The Raw, Percent, and Engineering Units options set the conversion rate of PV data values with values, such as monitoring, in the display area. Raw max and Raw min values specify PV data values, and EU max and EU min values specify values, such as monitoring, in the display area.

**Raw:** All Input/Output values to the connected device are shown in raw form with a conversion rate of 0.

When Raw is selected, set the values of Raw max, Raw min, EU max, and EU min as follows.

- Raw max=EU max
- Raw min=EU min

**Percent:** Values in percent are set in the display area.

When Percent is selected, set Raw max, Raw min, EU max, and EU min as follows.

- Raw max and Raw min values = user-defined value by the connected device
- EU max = 100
- EU min = 0

**Engineering Units:** Values of n mole fraction, defined by the user, are set in the display area.

When Engineering Units is selected, set Raw max, Raw min, EU max, and EU min as follows.

- Raw max and Raw min values = user defined value by the connected device
- EU max = n
- EU min = 0

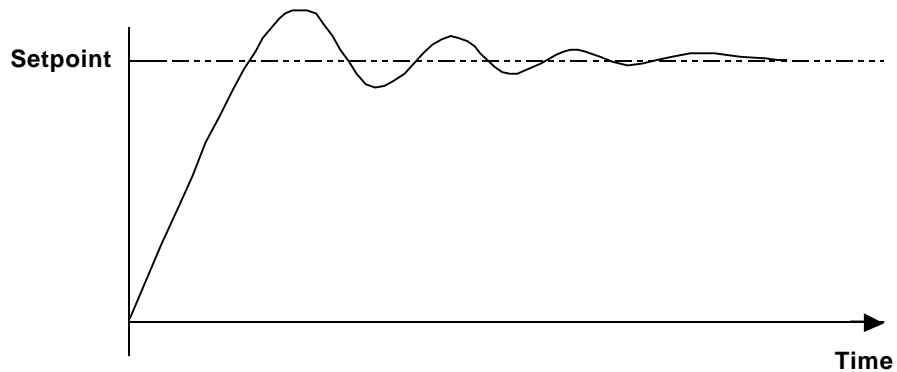
**Control**

Loop Update Time: Set the time cycle to acquire data from the connected device.

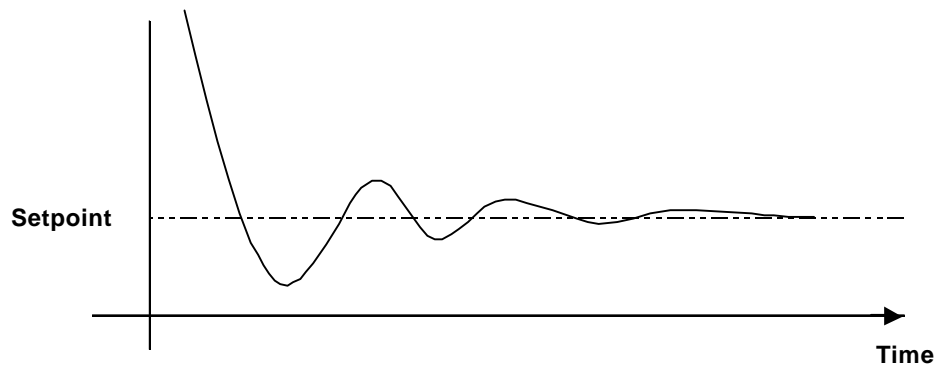
The Loop Update Time becomes the output update time/ period.

Setting the Sampling Time allows you to use the Filter feature, however, the Sampling Time should be set to a larger value than the Loop Update Time.

Direct (SP-PV): Specify to perform control to increase the control amount output when the Process Variable is smaller than the Setpoint (such as heater).



Reverse(PV-SV): Specify to perform control to decrease the control amount output when the Process Variable is greater than the Setpoint (such as cooler).



### Output Clamping

Sets the highest limit and the lowest limit of the Control Variable. When the Control Variable is outside this range, the highest limit or the lowest limit is output, and the status bit of Bit 3 or Bit 4 in Element [0] of the control block variable turns ON.

**Reference** For details, see • *Control Variable Exceeds the Upper Limit (Bit 3) or Lower Limit (Bit 4) in the ■ Control Block Variable section.*

### Anti-Reset Windup

Sets the highest and lowest limits of the Number of Integration Times per minute of Element [2] of the control block variable.

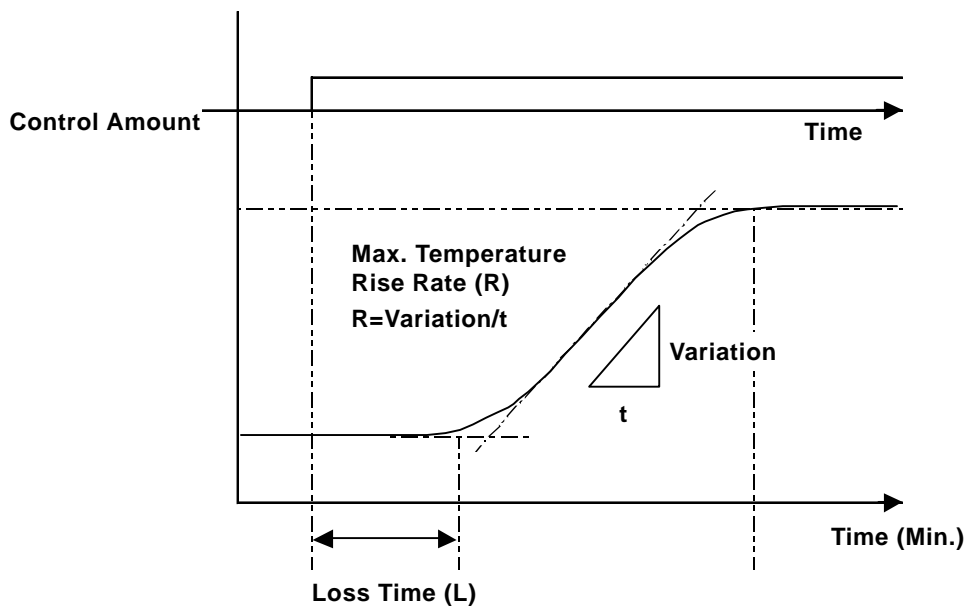
## ■ How to Adjust the PID Constant

This section explains how to adjust the PID constant using temperature control as an example. To obtain optimum PID control results, each constant of P (Proportional Element), I (Integral Element), and D (Derivative Element) has to be set to its optimum value. Step Response is one method of adjusting the PID constant against various control targets, and is based on temperature characteristics.

The optimum value might not be obtained with the Step Response method, depending on the control target used. In these cases, adjust the values in the Tune tab of the [PID] dialog box.

### Step Response

Step Response sets the Setpoint, and 100% of the control amount for the control target is output in steps. The following example, based on the chart of temperature characteristics, measures the maximum temperature slope (R) and the loss of time (L).





You can calculate constants of the proportional coefficient, the number of integral times, and the derivative time by substituting the measured values of the maximum temperature slope (R) and the loss time (L) in the following equation.

Please enter the calculated values in the Tune tab of the PID dialog box.

“Proportional Coefficient” =  $100/(0.83 \cdot R \cdot L)$  [%]

“Number of Integration Times” =  $1/(2 \cdot L)$  [cycles/min.]

“Derivative Time” =  $0.5 \cdot L$  [min.]

### 9.2.34 TON (Timer ON Delay)

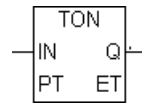
IN: Timer starting bit

PT: Preset time of timer

Q: Time up flag

ET: Present value of timer

Variable



When the timer input bit (IN) receives power, the TON instruction adds the preset time (PT), in milliseconds, and the timer output bit (Q) turns ON.

#### ◆ Overview

Special Variable	Description	Variable Type
Variable. PT	Preset Value	Integer
Variable. ET	Present Value	Integer
Variable. Q	Timer Output Bit	Discrete
Variable. TI	Timing Bit	Discrete

When power is passed to the timer starting bit (IN), the TON instruction starts, and:

- Variable.ET (the elapsed time) begins to increment in milliseconds.
- Variable.TI (the timing bit) turns ON.
- Variable.Q (the timer output bit) turns OFF.

When the elapsed time (Variable.ET) increments and equals the preset time (Variable.PT):

- Variable.ET (the elapsed time) holds the current value.
- Variable.TI (the timing bit) turns OFF.
- Variable.Q (the timer output bit) turns ON, and the instruction passes power.

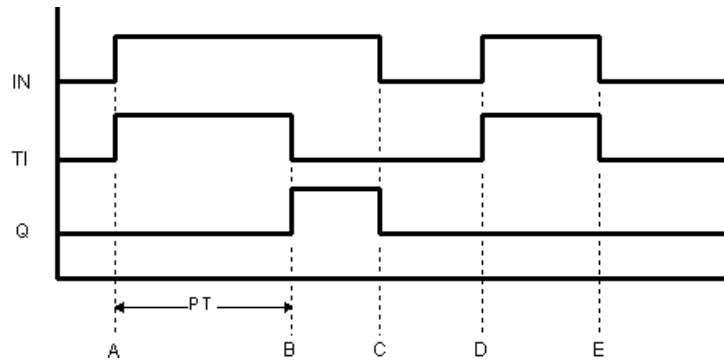
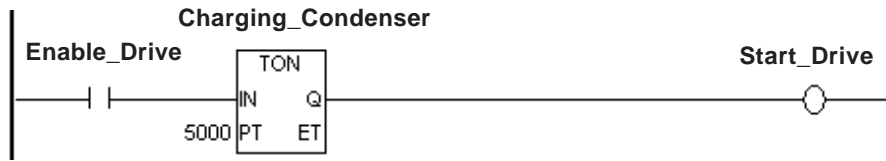
When the timer starting bit (IN) stops passing power to start the TON instruction:

- Variable.ET (the elapsed time) is reset to zero.
- Variable.TI (the timing bit) turns OFF.
- Variable.Q (the timer output bit) turns OFF.

## Chapter 9 – Instructions

### ◆ Example

In the following example, the drive will be started 5 seconds after "Enable\_Drive" is turned ON.



- A: When power is applied to the timer input bit (IN), the timing bit (TI) turns ON, the timer begins timing, and the elapsed time (ET) increments. The timer output bit (Q) remains OFF.
- B: The elapsed time (ET) equals the preset time (PT), the timer output bit (Q) turns ON, and the elapsed time (ET) stays fixed at the preset time. The timing bit (TI) turns OFF.
- C: The timer input bit (IN) turns OFF, the timer output bit (Q) turns OFF, and the elapsed time (ET) is reset to 0.
- D: The timer input bit (IN) turns ON, and the timing bit (TI) turns ON. The timer begins timing, and the elapsed time (ET) increments.
- E: The timer input bit (IN) is turned OFF before the elapsed time (ET) equals preset time (PT), the timer output bit (Q) remains OFF, the elapsed time (ET) is reset to 0.

### 9.2.35 TOF (Timer OFF Delay)

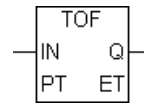
IN: Timer starting bit

PT: Preset time of timer

Q: Time up flag

ET: Present value of timer

Variable



When the timer input bit (IN) stops receiving power, the TOF instruction adds the preset time (PT), in milliseconds, and the timer output bit (Q) turns OFF.

#### ◆ Overview

Special Variable	Description	Variable Type
Variable. PT	Preset Value	Integer
Variable. ET	Present Value	Integer
Variable. Q	Timer output bit	Discrete
Variable. TI	Timing bit	Discrete

When power is passed to the timer starting bit (IN), the TOF instruction starts, and:

- Variable.ET (the elapsed time) is reset to zero.
- Variable.TI (the timing bit) turns OFF.
- Variable.Q (the timer output bit) turns ON, and the instruction passes power.

When the timer starting bit (IN) stops passing power to start the TOF instruction:

- Variable.ET (the elapsed time) begins to increment, in milliseconds.
- Variable.TI (the timing bit) turns ON.
- Variable.Q (the timer output bit) remains ON.

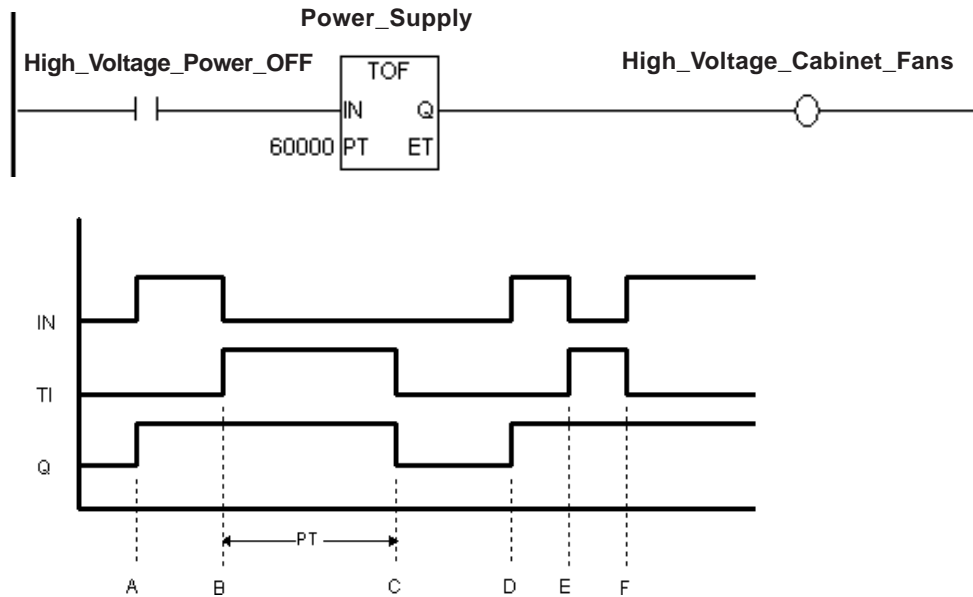
When the elapsed time (Variable.ET) increments and equals the preset time (Variable.PT):

- Variable.ET (the elapsed time) stays fixed at the preset value.
- Variable.TI (the timing bit) turns OFF.
- Variable.Q (the timer output bit) turns OFF.

## Chapter 9 – Instructions

### ◆ Example

The following diagram is an example of high-voltage cabinet fans that are kept running for 1 minute (60,000ms) after the high voltage turns OFF.



- A: The timer input bit (IN) turns ON, the timing bit (TI) remains OFF, the timer output bit (Q) turns ON, and the elapsed time (ET) is reset to 0.
- B: The timer input bit (IN) turns OFF, the timer starts timing (TI turns ON), and the timer output bit (Q) remains ON.
- C: When the elapsed time (ET) equals the preset time (PT), the timer output bit (Q) turns OFF, the timer stops timing (TI turns OFF), and the elapsed time stays fixed at preset time (ET=PT).
- D: The timer input bit (IN) turns ON, the timing bit (TI) remains OFF, the timer output bit (Q) turns ON, and the elapsed time (ET) is reset to 0.
- E: The timer input bit (IN) turns OFF, the timer starts timing (TI turns ON), and the timer output bit (Q) remains ON.
- F: Before the elapsed time (ET) equals the preset time (PT), the timer input bit (IN) turns ON, and the timer stops timing (TI turns OFF). The timer output bit (Q) remains ON, and the elapsed time (ET) is reset to 0.

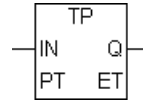
### 9.2.36 TP (Timer Pulse)

IN: Timer starting bit

PT: Preset time of timer

Q: Time up flag

ET: Present value of timer



When the timer input bit (IN) receives power one time, the TP instruction turns ON the output bit (Q) for the duration of the preset time (PT), in milliseconds.

#### ◆ Overview

Special Variable	Description	Variable Type
Variable. PT	Preset Value	Integer
Variable. ET	Present Value	Integer
Variable. Q	Timer output bit	Discrete
Variable. TI	Timing bit	Discrete

When power is passed to the timer starting bit (IN), the TP instruction starts, and:

- Variable.ET (the elapsed time) begins to increment in milliseconds.
- Variable.TI (the timing bit) turns ON.
- Variable.Q (the timer output bit) turns ON as the instruction passes power.

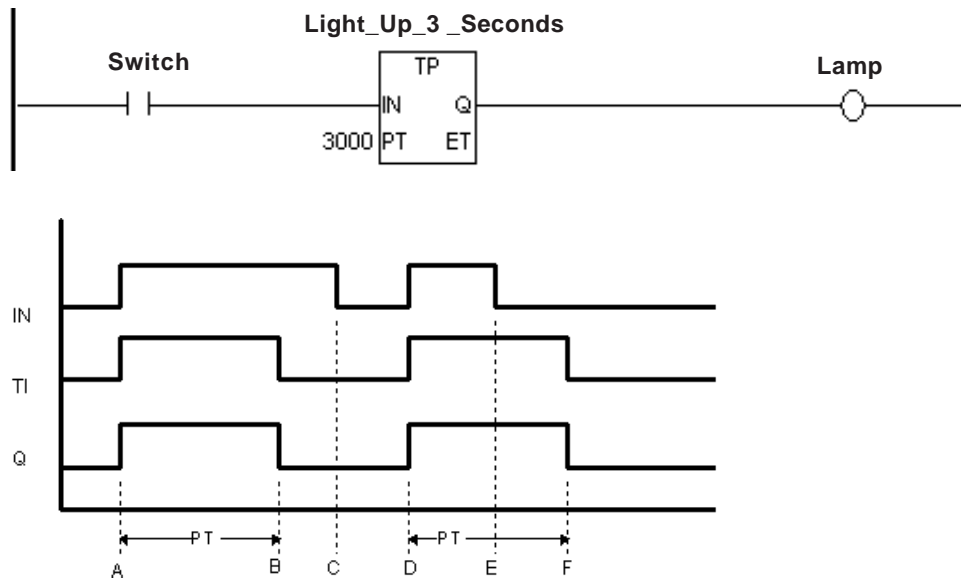
When the elapsed time (Variable.ET) equals the preset time (Variable.PT):

- Variable.ET (the elapsed time) stays fixed at the preset value if the TP instruction is still receiving power.
- Variable.ET (the elapsed time) resets immediately to zero if the instruction stops receiving power.
- Variable.TI (the timing bit) turns OFF.
- Variable.Q (the timer output bit) turns OFF.

When the timer starting bit (IN) stops passing power to start the TP instruction, the elapsed time (Variable.ET) is reset to zero, and the timer output bit (Variable.Q) turns OFF — only if it has already reached the value of the preset time (Variable.PT). Otherwise, it continues timing, and the timer output bit (Variable.Q) remains ON.

### ◆ Example

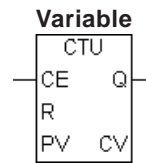
The following diagram is an example of a lamp that lights up for three seconds when the switch is pressed.



- A: The timer input bit (IN) turns ON, the timer starts timing (TI turns ON), and the timer output bit (Q) turns ON.
- B: When the elapsed time (ET) equals the preset time (PT), the timer output bit (Q) turns OFF, the timer stops timing (TI turns OFF), and the elapsed time stays fixed at the preset time (ET=PT).
- C: The timer input bit (IN) turns OFF, and the elapsed time (ET) is reset to 0.
- D: The timer input bit (IN) turns ON, the timer starts timing (TI turns ON), and the timer output bit (Q) turns ON.
- E: The timer input bit (IN) turns OFF, the timer continues timing (TI remains ON), and the timer output bit (Q) remains ON.
- F: When the elapsed time (ET) equals the preset time (PT), the timer output bit (Q) turns OFF, the timer stops timing (TI turns OFF), and since the timer input bit (IN) is OFF, the elapsed time (ET) is reset to 0.

**9.2.37 CTU (UP Counter)**

CE: Counter starting bit  
 R: Counter reset bit  
 PV: Preset value of counter  
 Q: Counter output  
 CV: Present value of counter



◆ **Overview**

Special Variable	Description	Variable Type
Variable.PV	Preset Value	Integer
Variable.CV	Current Value	Integer
Variable.R	Counter Reset	Discrete
Variable.UP	UP Counter	Discrete
Variable.QU	UP Counter Output	Discrete
Variable.QD	Down Counter Output	Discrete
Variable.Q	Counter Output	Discrete

When the counter input bit (CE) passes power, the current value (Variable.CV) is incremented by one if the counter reset bit (Variable.R) is OFF and the current value (Variable.CV) is smaller than Preset value (Variable.PV).

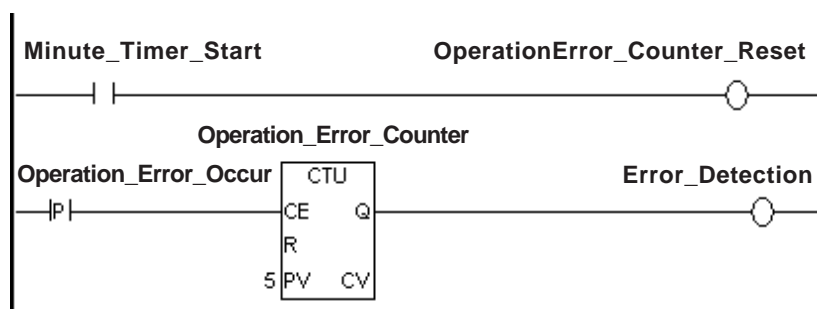
When the current value (Variable.CV) is equal to the preset value (Variable.PV), the counter output bit (Variable.Q) is turned ON, and the instruction passes power.

When the counter reset bit (Variable.R) is ON, the current value (Variable.CV) is reset to zero.

The counter output bit (Variable.Q) is also turned OFF.

◆ **Example**

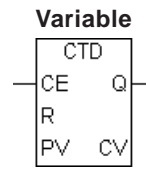
The following diagram is an example of the CTU instruction notifying the Error\_Detection output when five errors have been counted during a one-minute period.



**Note:** The counter is reset every scan. To count an event like the example above, be sure that the PT instruction is positioned before the CTU instruction's position. The CTU instruction is a level input.

**9.2.38 CTD (DOWN Counter)**

- CE: Counter starting bit
- R: Counter reset bit
- PV: Preset value of counter
- Q: Counter output
- CV: Present value of counter



◆ **Overview**

Special Variable	Description	Variable Type
Variable. PV	Preset Value	Integer
Variable. CV	Current Value	Integer
Variable. R	Counter Reset	Discrete
Variable. UP	UP Counter	Discrete
Variable. QU	UP Counter Output	Discrete
Variable. QD	Down Counter Output	Discrete
Variable. Q	Counter Output	Discrete

When the counter input bit (CE) passes power, the current value (Variable.CV) is decremented by one if the counter reset bit (Variable.R) is OFF.

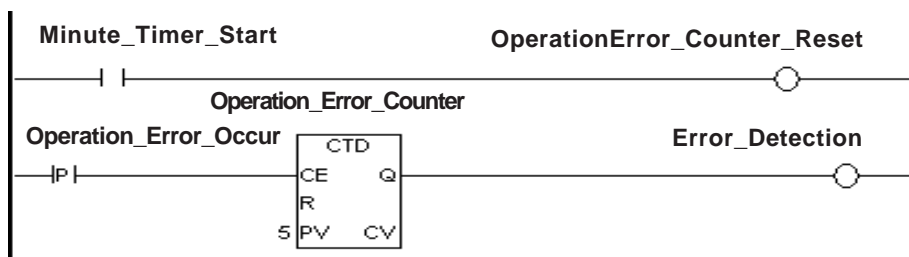
When the current value (Variable.CV) becomes equal to or less than zero after decrementing, the counter output bit (Variable.Q) is turned ON, and the instruction passes power.

When the counter reset bit (Variable.R) is ON, the preset value (Variable.PV) is set to the current value (Variable.CV).

The counter output bit (Variable.Q) is also turned OFF.

◆ **Example**

The following diagram is an example of the CTD instruction passing power and notifying the Error\_Detection output when five errors have been counted during a one-minute period. The timer resets the counter every minute.



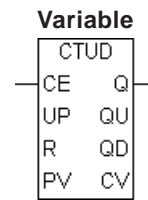
**Note:**

The counter is reset every scan. To count an event like the example above, be sure that the PT instruction is positioned before the CTU instruction's position. The CTD instruction is a level input.



### 9.2.39 CTUD (UP/DOWN Counter)

CE: Counter starting bit  
 UP: Counter Up Instruction  
 R: Counter reset bit  
 PV: Preset value of counter  
 Q: Counter output  
 QU: UP Counter flag  
 QD: Down Counter flag  
 CV: Present value of counter



#### ◆ Overview

Special Variable	Description	Variable Type
Variable. PV	Preset Value	Integer
Variable. CV	Current Value	Integer
Variable. R	Counter Reset	Discrete
Variable. UP	UP Counter	Discrete
Variable. QU	UP Counter Output	Discrete
Variable. QD	Down Counter Output	Discrete
Variable. Q	Counter Output	Discrete

When executing the CTUD instruction while the counter up instruction Variable.UP is ON, the execution is similar with the CTU instruction (up-counter).

When Variable.UP is OFF, the execution is similar with the CTD (down-counter) instruction.

After executing the CTUD instruction:

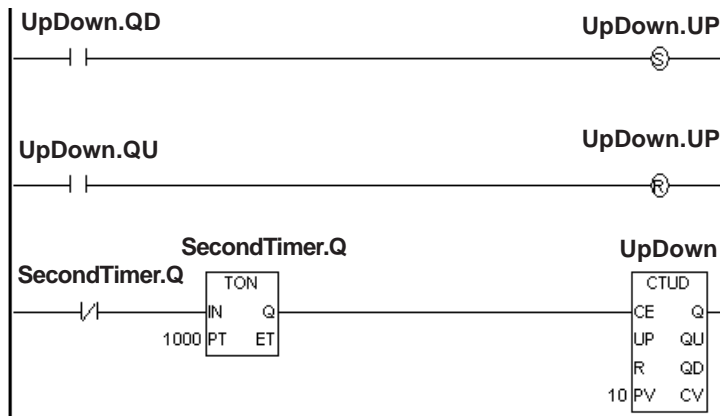
- If the current value (Variable.CV) is equal to or greater than the preset value (Variable.PV), the Counter Output and UP Counter Output (Variable.Q and Variable.QU) are turned ON.
- If the current value (Variable.CV) is equal to or less than zero, the Counter Output and Down Counter Output (Variable.Q and Variable.QD) are turned ON.

#### ◆ Example

The following diagram is an example of the CTUD instruction continuously counting up, from 0 to 10, and then down from 10 to 0.

The SecondTimer outputs a pulse to the Up/Down Counter every second.

The UP bit turns ON when the Up/Down Counter reaches 0, and turns OFF when the Up/Down counter reaches 10 (the preset value).



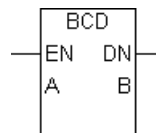
**Note:**

If the counter reset bit (Variable.R) turns ON when the Counter Up instruction (Variable.UP) is ON, the current value (Variable.CV) is set to zero. If the counter reset bit (Variable.R) turns ON when the Counter Up instruction (Variable.UP) is OFF, the preset value (Variable.PV) is entered to the current value (Variable.CV).

### 9.2.40 BCD (BCD Conversion)

A: Data

B: Result to be stored



When the BCD instruction is executed, a binary number assigned to A is converted to binary-coded decimal format, and the result is placed in B.

The BCD instruction does not pass power if an error occurs. The following table lists the combinations of A and B in which BCD instructions can be executed.

A	B
Integer	Integer
Integer Constant	

The largest value of A that can be converted is 0 x 5F5E0FF. If A is too large, #FaultCode is updated with the error code, and #Overflow is turned ON.

**Reference** 8.2.15 #Faultcode and 8.2.18 #Overflow



**Note:**

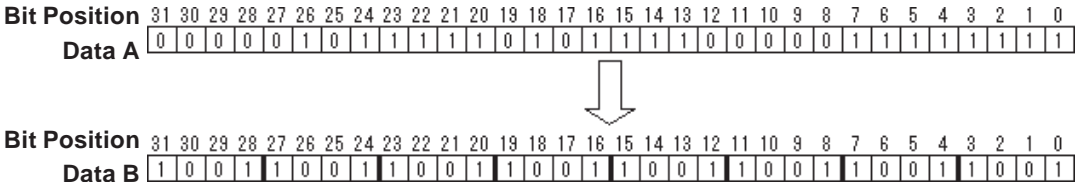
If the value cannot be converted, the value in B is undefined.

◆ Example

When Start is turned ON, Data A is converted to BCD and stored in Data B.

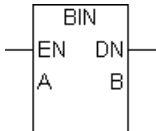


Example) BIN data "99999999" is designated for data A, and BCD conversion is performed.



**9.2.41 BIN (Binary Conversion)**

A: Data  
B: Result to be stored



When the BIN instruction is executed, a binary coded decimal number assigned to A is converted to binary format, and the result is placed in B.

The BIN instruction does not pass power if an error occurs. The following table lists the combinations of A and B in which BIN instructions can be executed.

A	B
Integer	Integer
Integer Constant	

If A is not a valid BCD number, #FaultCode will be updated with the error code, and #Overflow will turn ON.

▼ **Reference** ▲ 8.2.15 #Faultcode and 8.2.18 #Overflow



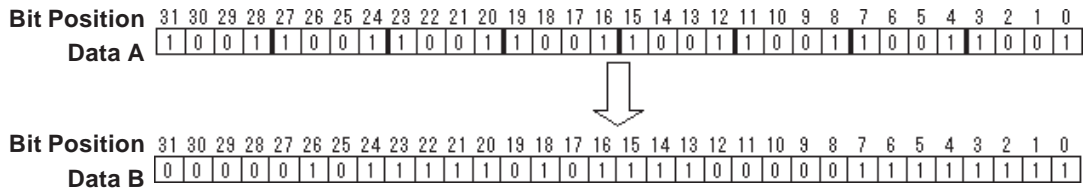
**Note:** If the value cannot be converted, the value in B is undefined.

◆ Example

When Start is turned ON, Data A is converted to BIN and stored in Data B.



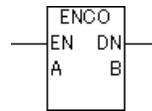
Example) BIN data "99999999" is designated for data A, and BCD conversion is performed.



**9.2.42 ENCO (Encode)**

A: Data

B: Result to be stored



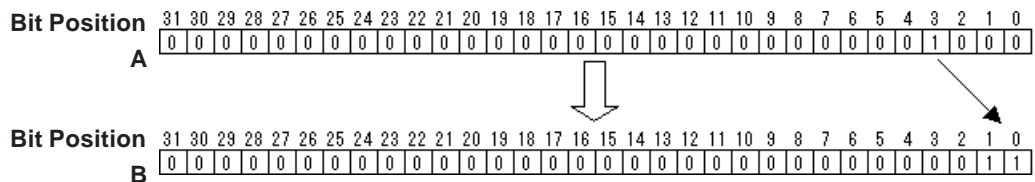
The value entered in A is encoded and output to B. The ENCO instruction reads the 32 bits in A for the bit position that is ON, and this position is output to B as a binary value. If several bits in A are ON, the most significant bit position is output to B.

The ENCO instruction always passes power.

The combinations of valid variable data types for the ENCO instruction are as follows:

A	B
Integer	Integer
Integer Array	Integer Array (same size as A)
Integer Constant	Integer

E.g.: If 0x00000008 is entered in A, the output B is 0x00000003.



**Note:**

- If 0 is entered in Input A, the error code “13” is set to #FaultCode as a minor error (#Overflow).

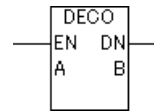
**Reference** 8.2.19 #Overflow

- The ENCO instruction does not support variable modifiers (assigned bit, word, or byte).

**9.2.43 DECO (Decode)**

A: Data

B: Result to be stored



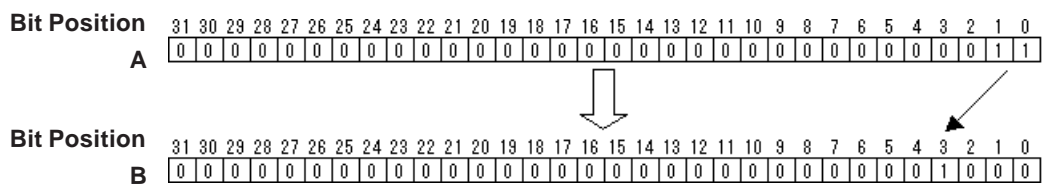
The value entered in A is decoded and output to B. The DECO instruction reads A as a binary value, and the corresponding bit position in B is up. 0 to 31 are available for input.

The DECO instruction always passes power.

The combinations of valid variable data types for the DECO instruction are as follows:

A	B
Integer	Integer
Integer Array	Integer Array (same size as A)
Integer Constant	Integer

E.g.: If 0x00000003 is entered in A, the output B is 0x00000008.



**Note:**

- If a value other than 0 to 31 is entered in Input A, the error code “13” is set to #FaultCode as a minor error (#OverFlow).

**Reference** 8.2.19 #Overflow

- The DECO instruction does not support variable modifiers (assigned bit, word, or byte).

**9.2.44 JMP (Jump)**

—>> LabelName

When the JMP instruction receives power, control jumps to the specified label. Unlike the JSR instruction, control does not automatically return to the rung following the JMP rung.

A jump cannot be made over a START, SUB START , SUB END, ACT START or ACT END label.

Jumping upward can create an infinite loop.

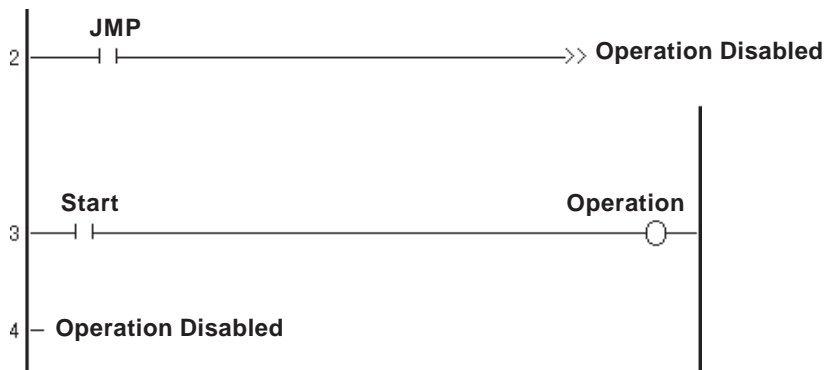


**Note:** Be sure that the time required to execute the entire program will not exceed the value of the Watch Dog Timer.

**Reference** 8.2.26 #WatchDogTime

◆ **Example**

If the Jump Instruction is ON, rung 3's instruction will be skipped and not executed. Control will jump to rung 4 with the label "Operation Disabled", and instructions below rung 4 will be executed.



### 9.2.45 JSR (Jump Subroutine)

—>>SubroutineName<<

When the JSR instruction receives power, the control jumps to the specified subroutine. After the subroutine executes, control returns to the rung that follows the JSR instruction and continues to execute that rung's instruction. A subroutine name can not be duplicated.

JSR must be the last instruction on a rung.

#### ◆ Restrictions

A maximum of 128 subroutine jumps from a subroutine can be executed.



**Note:**

Be sure that the time required to execute the entire program will not exceed the value of the Watch Dog Timer.

▼ **Reference** ▲ 8.2.26 #WatchDogTime

### 9.2.46 RET (Return Subroutine)

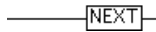
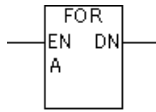
—<RETURN>—|

When the RET instruction receives power, control is forced from a subroutine and is returned to its original location. Execution continues from the rung that follows the Jump Subroutine (JSR) instruction.

When a subroutine is completed, the SUB END instruction forces the program to automatically return to the jump point. As a result, the RET instruction is not always needed to perform this function.

The RET instruction must be the last instruction on a rung.

## 9.2.47 FOR/NEXT (Repeat)



The FOR/NEXT instruction repeats the logic program between corresponding FOR and NEXT instructions, for the number of times specified in A. After executing the logic program between FOR and NEXT the specified number of times (A), the step that follows the NEXT instruction will be processed.

If A is equal to or less than 0, the logic program between FOR and NEXT is not executed, but jumps to the step that follows the NEXT instruction.

The FOR/NEXT instruction always passes power.

Valid variable data types for the FOR/NEXT instruction are as follows:

A
Integer
Integer Array
Integer Constant

### ◆ Restrictions

- Each FOR instruction requires a NEXT instruction.
- Do not insert instructions before or after FOR and NEXT instructions on the same rung.
- Up to 64 nests can be included in each instruction.

If the instruction exceeds more than 64 nests, a major error occurs and error code “4” is displayed in #FaultCode.

- Two (2) stacks are used for one nesting. The total number of stacks that can be used in a logic program is 128. The only other instruction that uses stacks is the JSR instruction, which uses one (1) stack.

▼ **Reference** ▲ 9.2.44 JSR (Jump Subroutine)



#### Note:

- For information about the errors or warnings displayed by the Editor’s error check, refer to the Pro-Control Editor Operation Manual, Chapter 7, Appendix 1 – “Errors and Warnings.”

▼ **Reference** ▲ 8.2.16 #FaultCode

- When specifying the number of nests, the time required for the program’s entire execution must NOT exceed the value of Watchdog Timer.

▼ **Reference** ▲ 8.2.27 #WatchdogTime



# 10 LS Area Refresh

## 10.1 LS Area Refresh Overview

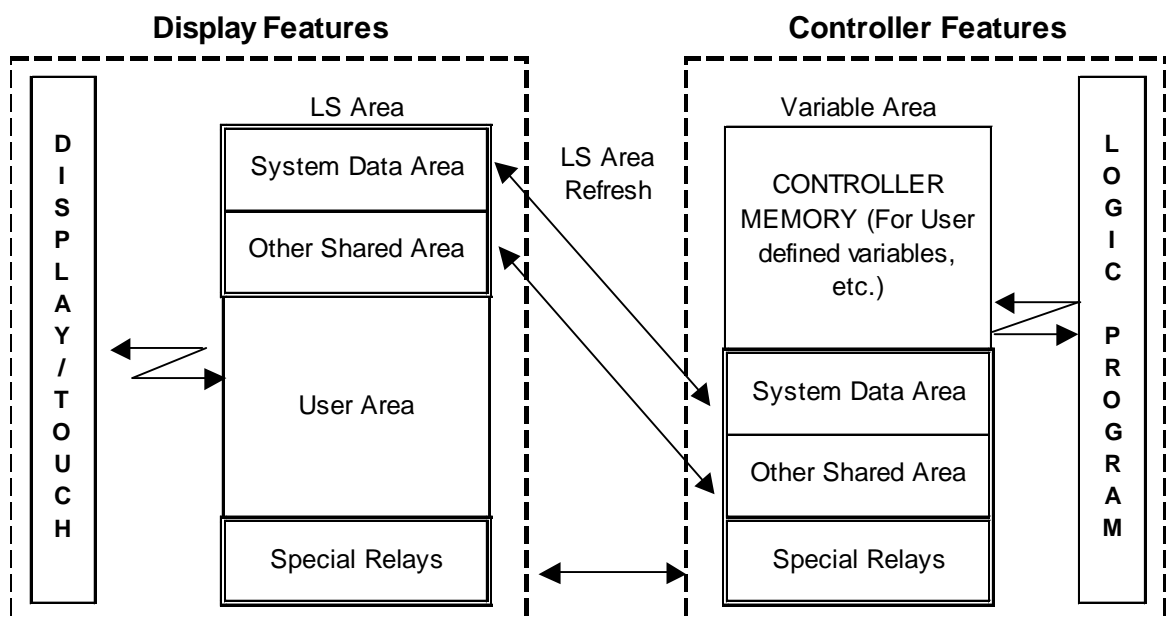
### ■ LS Area Refresh Feature

The LT unit uses the LS Area's System Data Area to control the changing of screens, the sounding of buzzers, etc. These are processed as LT display features.

Therefore, when using the functions assigned to the System Data Area such as the screen change and clock function via the Controller Feature, the data in the LS Area should be shared between the Display Features and Controller Features through registering the LS Area as variables.

This is defined as the LS Area Refresh.

It is also possible to use an area outside of the System Data Area if the LT controller features or display features need to share data.



The LS area refresh timing and the Logic Symbol update timing are not synchronized. Therefore be sure to use an interlock when programming a refresh for either of these in your ladder logic program.

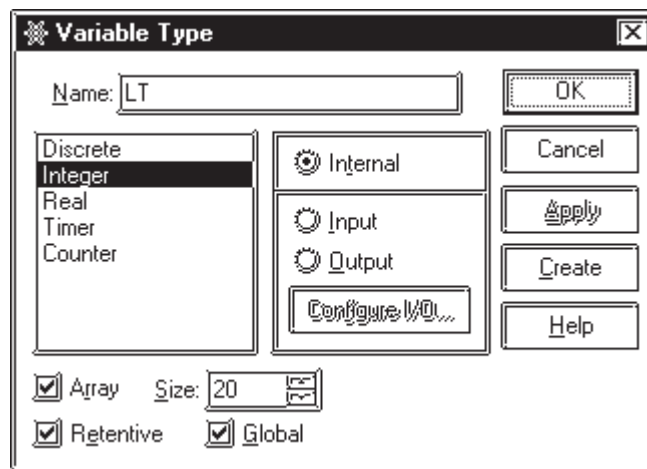
## 10.2 LS Area Refresh Settings

When using the logic program to designate the LS Area, the desired variable must first be registered in LT Editor. This section describes this procedure.

### ■ Variable Registration

In LT Editor's Data menu, click Variable Type to open the Variable Type dialog box. This section describes how to register a variable with a variable name "LS" as an internal integer array.

The size should be calculated by adding the number of words of data to be shared to the System Area's 20 words. (Example: When sharing 16 words of data with the System Data Area, enter "36" words, i.e., 20 words for the System Data Area plus 16 words.)



- Note:**
- The Special Relay Area is called the LSS area.
  - The maximum LS size is 276 words.

The relationship between variables and addresses are listed in the following table.

Variable Name*1	Address	LS Address	
LS[0]	0	LS0000	} System Data Area
LS[1]	1	LS0001	
:	:	:	
LS[19]	19	LS0019	
:	:	:	} Other Shared Data
LS[275]	275	LS0275	
LSS[0]	2032	LS2032	} Special Relays
LSS[1]	2033	LS2033	
:	:	:	
LSS[15]	2047	LS2047	

1. Variable Name: System Variables managed by the LT's ladder logic program

**Reference** For details about the LT Type C unit, refer to the *Device/PLC Connection Manual* (provided with the LT Editor).

## 10.2.1 LS Area - When not using a Device/PLC



**Do NOT use any areas designated as Reserved.**

### ■ System Data Area

You can use controller features in your logic program to update this area and control the "LT Screen Change" and "Backlight ON/OFF".



**This area can be accessed by registering LT Editor's internal "[LS]" integer array variables.**

Address	Var. <sup>*1</sup> Name	Detail	Function	Bit	Particulars
1	LS[1]	Status		0, 1	Reserved
				2	Now Printing
				3	Writes a set value
				4 to 9	Reserved
				10	Backlight Burnout Detection
				11 to 15	Reserved
2	LS[2]	Error Status  Each bit changes according to the GP error function. When an error occurs, the corresponding bit will turn on.		0, 1	Unused
				2	System ROM/RAM
				3	Screen Memory Checksum
				4	SIO Framing
				5	SIO Parity
				6	SIO Overrun
3	LS[3]	A bit that has turned ON remains ON until the power is turned OFF and back ON, or until RUN mode is re-entered from OFFLINE mode.		7, 8	Unused
				9	Initialization of Internal Memory Checksum Necessary
				10	Timer Lock Error
				11 to 15	Unused
4	LS[4]	Clock Data (Year)	"Year / Month / Day / Hour / Minute" Data is stored in BCD's 2digits. (E.g.) 98/02/01 17:15	0 to 7	Stores the last 2 digits of the Calendar year
		8 to 15		Unused	
5	LS[5]	Clock Data (Month)		0 to 7	Stores 01 to 12 (Month) as 2 BCD digits
				8 to 15	Unused
6	LS[6]	Clock Data (Day)		0 to 7	Stores 00 to 31 (Day) as 2 BCD digits
				8 to 15	Unused
7	LS[7]	Clock Data (Hour)		0 to 7	Stores 00 to 23 (Hour) as 2 BCD digits
				8 to 15	Unused
8	LS[8]	Clock Data (Minute)		0 to 7	Stores 00 to 59 (Minute) as 2 BCD digits
				8 to 15	Unused
10	LS[10]	Interrupt Output (Touch OFF)	If you use a Switch Part to write in word data, the bottom 8 bits will be output as an interrupt code after Touch OFF. FFh will not be output.		

\*1 Variable names used when accessing the LT.

## Chapter 10 – LS Area Refresh

LS Address	Var. *1 Name	Detail	Function	Bit	Particulars
11	LS[11]	Control		0	Backlight
				1	Buzzer ON
				2	Starts Printing
				3	Reserved
				4	Buzzer - - - 0:enabled 1: disabled
				5	Reserved
				6	Interrupt Output when touching panel to turn the display ON. (Interrupt Code:FFh) 0: Disabled 1: Enabled
				7 to 10	Reserved
				11	Hard copy output - 0: Enabled 1: Disabled
12 to 15	Reserved				
12	LS[12]	Screen Display ON/OFF	FFFFh : Screen clears almost immediately 0h: Screen turns ON		
13	LS[13]	Interrupt Output	Using a Switch Part or other method to write absolute value data from LT causes an output of the interrupt code using the contents of the bottom 8 bits ( Will not output FFh)		
15	LS[15]	Screen Display No.	Write the Screen No. in binary to change the screen display	0 to 14	Screen change number, 1 to 8999. ( 1 to 1999 when using BCD input)
				15	Forced Screen Change 0: Normal 1: Forced Screen Change
16	LS[16]	Reserved			
17	LS[17]				
18	LS[18]				
19	LS[19]				

\*1 Variable names used when accessing the LT.

## ■ Special Relay



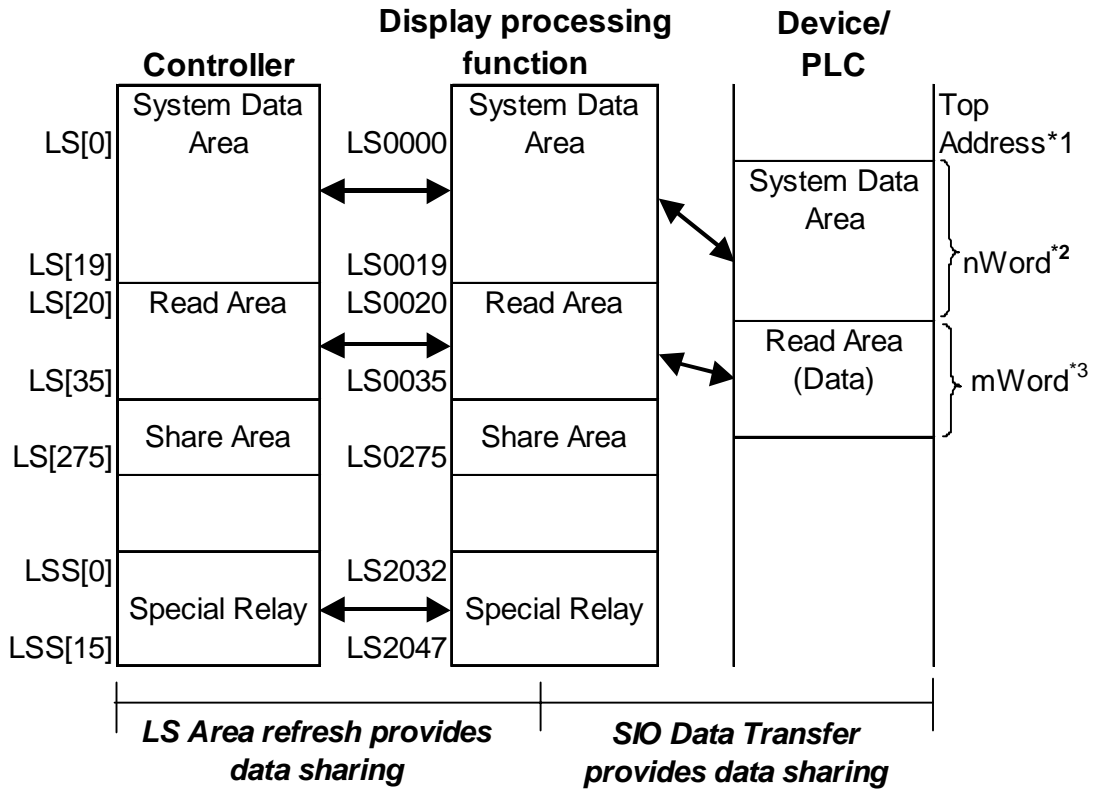
**This area can be accessed by registering LT Editor's internal "[LSS]" integer array.**

LS Address	Var. Name <sup>*1</sup>	Contents
2032	LSS [0]	Share Relay Data
2033	LSS [1]	Reserved
2034	LSS [2]	
2035	LSS [3]	Binary Counter - 1 second
2036	LSS [4]	Part Scan Time
2037	LSS [5]	Reserved
2038	LSS [6]	Part Scan Counter
2039	LSS [7]	Reserved
2040	LSS [8]	
2041	LSS [9]	
2042	LSS [10]	
2043	LSS [11]	
2044	LSS [12]	
2045	LSS [13]	
2046	LSS [14]	
2047	LSS [15]	

*\*1 Variable names used when accessing the LT.*

## 10.3 LT and Device/PLC Data Sharing

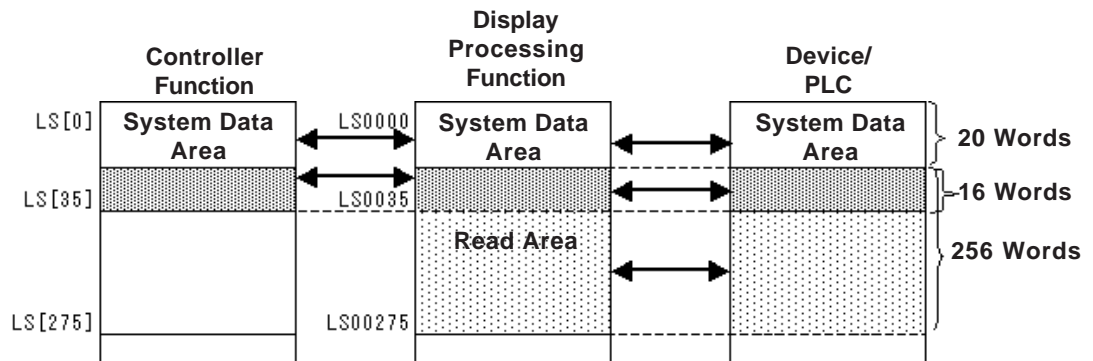
When using controller features to communicate with a Device/PLC, data is shared via the LS Area. However, if data sharing between the controller functions and the Device/PLC's data register exceeds 16 words, the performance of screen display functions may slow.



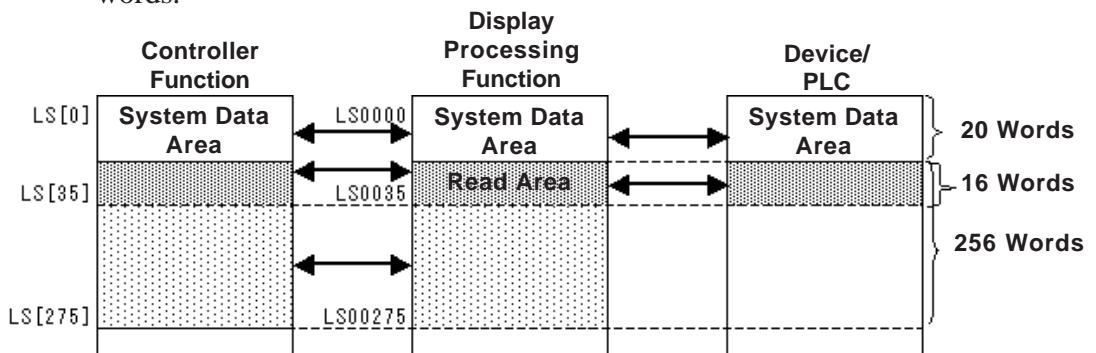
1. Start Address defined in Initial Settings.
2.  $n = 0$  to 20 Depends on the System Data Area setting items selected in Initial Settings .
3.  $m = 0$  to 16 Depends on size of Read Area designated in Initial Settings.

If you want set the Read Area and Variable LS to exceed 16 words, the Read Area can be set up to 256 words, and Variable LS can be set up to 276 words. A maximum of 16 words is recommended when setting data that is shared between the controller, display processing function and Device/PLC.

E.g.: When the Variable LS size is set to 36 words and the Read Area is set to 256 words.



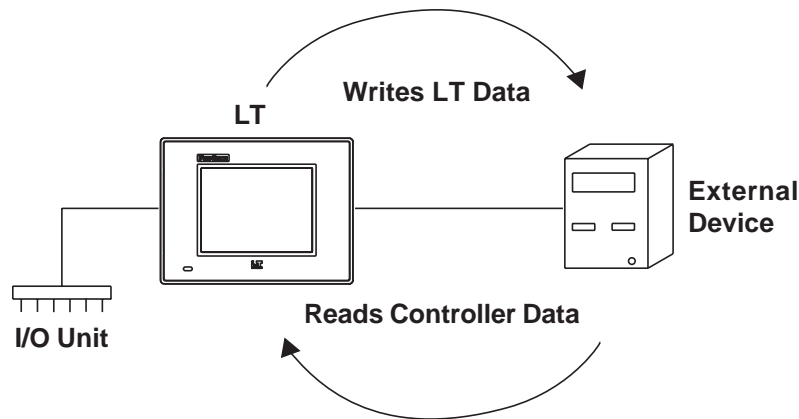
E.g.: When the Variable LS size is set to 276 words and the Read Area is set to 16 words.



- **When the controller's logic program, tags used to update the Display Processing feature and the logic program from an external I/O unit attempt to change the same variable at same time, priority is determined by the timing.**
- **When writing data to the LT unit's Read Area, be sure that the Write Via Parts and Write Via Logic Program functions of the Controller do not conflict.**



**Note:** Utilizing the Read Area to share data between the LT and Device/PLCs allows you to use the LT as the expansion unit of an external device as well as for construction of POP machines for factory automation or an I/O information terminal for production control.



### 10.3.1 LT Type C and Device/PLC LS Area Refresh Cautions

Use the LS Area Refresh feature to control the system area using the controller feature or to view Read Data from a Device/PLC. Pro-face recommends that you use the data send/receive related Initialize area or the Operation Designation Change parameter settings to control the refreshing of data in this area, rather than refreshing the data in addresses LS000 to LS0035 and LS2032 to LS2047 intermittently via the controller feature.

If the frequency of the LS Area's data refresh is increased, the LS Area Refresh may not be executed within one scan. As a result, Device/PLC communication errors may occur.

Variable LS is an integer variable, and the length is 32 bits.

When the System Data Area is 16 bits long, the low 16 bits are enabled.



# 11 I/O Drivers

This chapter explains I/O drivers related to the LT unit's I/O unit.

▼ **Reference** ▲ *When using an LT Type-H unit, refer to the **LT Type-H Driver Manual**.*

## 11.1 I/O Driver Overview

To perform external I/O, the LT unit's I/O unit must be attached and its related I/O drivers must be installed.



**Note:**

When an I/O error occurs and the controller stops, create the following logic program. There will be a delay of approximately one scan, from the time the error is detected until the time the logic program stops.

In the following example, an I/O error is detected with #IOFault, and logic execution is stopped by assigning 1 to #Command.



When an I/O error occurs, #IOFault will turn ON. Detailed information can be checked by #IOStatus.

▼ **Reference** ▲ *8.2.17 #IOFault and 8.2.19 #Command*

## 11.2 Flex Network I/F Driver

This section describes the Flex Network driver menus in the LT unit's OFFLINE mode. Prior to executing any Flex Network Driver menu instructions, be sure to download the Flex Network driver from LT Editor software in your PC. The Flex Network Driver is used with the LT Type-B, Type-B+, and Type-C.

**Reference** For the procedures on shifting to OFFLINE mode, refer to the *LT Series User Manual* (sold separately).

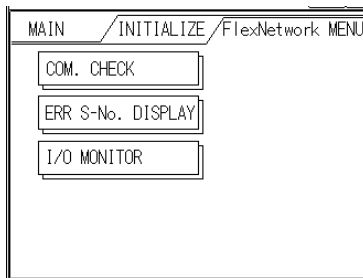
### 11.2.1 Flex Network I/F Unit Self-Diagnosis

This section describes how to operate the self-diagnostics of the Flex Network I/F unit.

**Reference** For details on the self-diagnostics of the LT main unit, refer to the *LT Series User Manual* (sold separately).

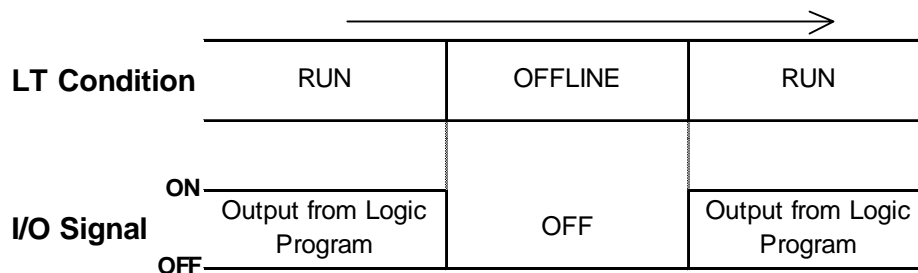
Select [FLEX NETWORK DRIVER] in the [CONTROLLER MENU]. The following window will then appear.

<To select communication check>



Important

The following diagram describes the action of the LT and I/O signal when the RUN mode of a ladder logic program is reset or shifted to OFFLINE mode. All I/O unit output hold settings will be ignored. Be sure you understand these actions before performing a reset or shifting to OFFLINE mode.



- The RESET mode's I/O signal OFF timing is NOT fixed.

## 11.2.2 Communication Check

The number of the Flex Network I/O units that have been connected to the Flex Network I/F units, as well as the S-No.S that have been connected to each I/O unit, will be checked.

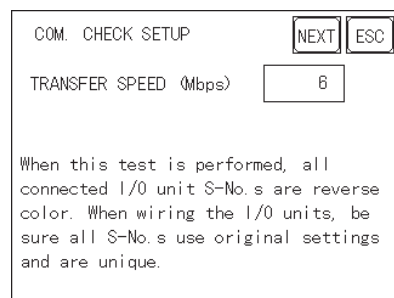
Via the communication check operation, the following items can be checked:

- currently connected I/O units
- currently malfunctioning I/O units (connection section)

### ◆ Communication Check Procedure

1. Press the COMMUNICATION CHECK button, and the COMMUNICATION CHECK SETUP window will appear.

Set Communication Speed to either 6 or 12. Setting the communication speed faster may cause the unit to be easily influenced by noise. Generally, set this speed to 6Mbps.



COM. CHECK SETUP

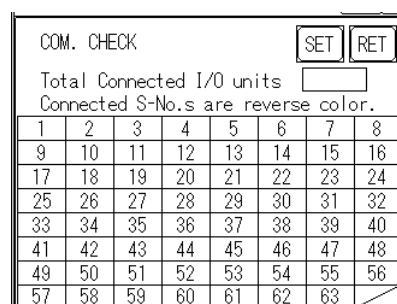
TRANSFER SPEED (Mbps) 6

When this test is performed, all connected I/O unit S-No.s are reverse color. When wiring the I/O units, be sure all S-No.s use original settings and are unique.

2. Press the NEXT button, and the COMMUNICATION CHECK window will appear.

Press START to begin the communication check.

The currently connected I/O unit's S-No. will be displayed in reverse color.



COM. CHECK

Total Connected I/O units 8

Connected S-No.s are reverse color.

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	

To return to the FLEX NETWORK MENU window, press the RET button.

**11.2.3 Error S-No.**

If the Error Code No. 841 occurs while a logic program is being executed, the S-Nos. of the I/O units that have been excluded from the communication circuit and malfunctioning I/O units will be checked.

**Reference** 11.3.3 *Flex Network I/F Unit Troubleshooting*

**◆ Error S-No. Procedure**

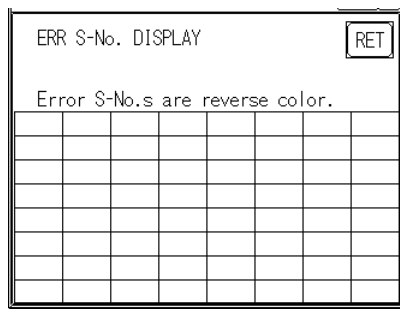
1. Touch the CONTROLLER MENU window's FLEX NETWORK DRIVER selection.

The FLEX NETWORK DRIVER MENU will appear.

2. Press the FLEX NETWORK DRIVER MENU's ERROR S-NO. DISPLAY.

The ERROR S-NO. DISPLAY window will appear, and the error check will begin.

The currently connected I/O unit's S-Nos. will appear, and the I/O unit S-No. with the error will be shown in reverse color.

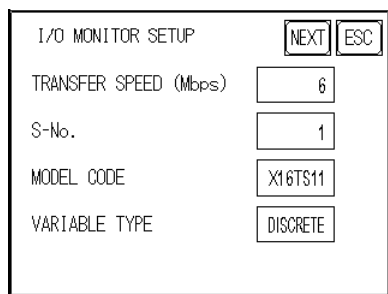


**11.2.4 I/O Monitor (I/O Connection Check)**

Check each Input and Output terminal between the LT and I/O unit. To check inputs, monitor the I/O unit of output signals on the LT. To check outputs, monitor the LT unit's output signals on the I/O unit.

**■ I/O Monitor Check Procedure**

1. Select the CONTROLLER MENU window's FLEX NETWORK DRIVER, and the FLEX NETWORK DRIVER MENU will appear.
2. Select the FLEX NETWORK DRIVER MENU window's I/O MONITOR, and the following I/O MONITOR SETUP window will appear.



**TRANSFER SPEED**

Set TRANSFER SPEED to either 6 or 12 Mbps. Setting a faster transfer speed may result in interference caused by noise. Normally, set this speed to 6Mbps.

**S-No. (Station no.)**

Select S-No. from 1 to 63.

**MODEL CODE**

Select one of the following models: X16TS, Y08RL, Y16SK, Y16SC, XY08TS, AD04AH, and DA04AH.

The FN-X32TS, FN-XY16SK, FN-XY16SC, FN-XY32SK, and FN-XY32SC models are not included in the selection. Therefore, select a substitute model, from the table below, that can check the I/O monitor's connection.

Models Performing I/O Monitoring		FN-X32TS	FN-XY16SK FN-XY16SC	FN-XY32SKS*1 FN-XY32SCS*1		
Substitute Models Performing I/O Monitoring		X16TS	X16TS Y16SK or Y16SC	XY08TS		
S-No.	+0	Input	0-15	0-15	–	0-7
		Output	–	–	0-15	0-7
	+1	Input	16-31	–	–	8-15
		Output	–	–	–	8-15
	+2	Input	–	–	–	16-23
		Output	–	–	–	16-23
	+3	Input	–	–	–	24-31
		Output	–	–	–	24-31

◆ **Monitoring the FN-X32TS**

Use X16TS as a substitute.

Lower 16 bits (0-15 bits) can be monitored by assigning the station number set in the I/O unit to the S-No.

Upper 16 bits (16-31 bits) can be monitored by assigning values created by adding 1 to the station number set in the I/O unit to the S-No.

◆ **Monitoring the FN-XY16SK or the FN-XY16SC**

Use X16TS as a substitute for input, and Y16SK or Y16SC as a substitute for output.

Input and Output cannot be monitored simultaneously.

◆ **Monitoring the FN-XY32SK, FN-XY32SC**

Use XY08TS as a substitute.

Input and output of bits 0–7 can be monitored by assigning the station number set in the I/O unit to the S-No.

Input and output of bits 8–15 can be monitored by assigning the values by adding 1 to the station number set in the I/O unit to the S-No.

Input and output of bits 16–23 can be monitored by assigning to the S-No. the values created by adding 2 to the station number set in the I/O unit.

Input and output of bits 24–31 can be monitored by assigning to the S-No. the values created by adding 3 to the station number set in the I/O unit.

**VARIABLE TYPE**

Select either DISCRETE or WORD.

\* Only the Word setting can be used for FN-AD04AH and FN-DA04AH.

3. Press the NEXT button, and the following I/O MONITOR window will appear.

This window’s items will vary depending on the VARIABLE TYPE selected.

**Reference** Refer to the information for the corresponding I/O unit model(s).



**Note:** This I/O monitor cannot be used with the high-speed counter and single-axis positioning unit.

◆ **FN-X16TS/FN-XY08TS/FN-Y08RL/FN-Y16SK/FN-Y16SC/FN-XY16SK/FN-XY16SC/FN-X32TS/FN-XY32SK/FN-XY32SC**

**I/O MONITOR (when VARIABLE TYPE is set to DISCRETE)**

The INPUT area terminal numbers where data has been entered will appear in reverse color. Touching an Output area terminal number will output the data and reverse that number’s color.

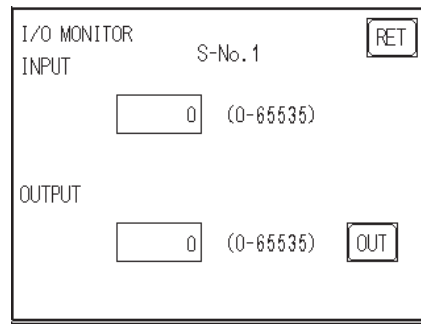
I/O MONITOR								RET
INPUT								S-No.1
0	1	2	3	4	5	6	7	
8	9	10	11	12	13	14	15	
OUTPUT								
0	1	2	3	4	5	6	7	
8	9	10	11	12	13	14	15	

The above windows display the maximum input/output points of an I/O unit in the Flex Network system. The number of input/output points depends on each I/O unit model. Use the range of I/O points within each unit, beginning with “0.”

When using an input-only I/O unit, use only the input area of the window, and when using an output-only I/O unit, use only the output area. When using a unit with inputs and outputs, use both the input and output areas.

**I/O MONITOR (when the VARIABLE TYPE is set to WORD)**

The input data, if any, will be displayed in the input field. Enter the necessary data in the output section via the ten-key keypad. After entering data, touch the OUT key to output the data. Data will be displayed in the decimal format.

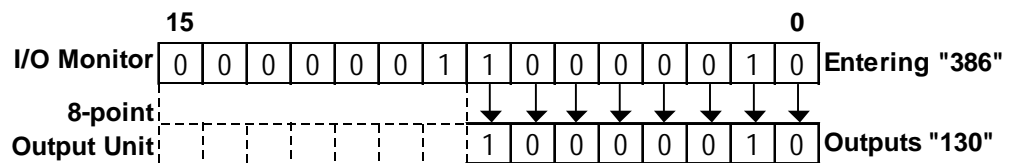


- **Enter data within the output range, according to the number of the I/O points in each I/O unit.**

I/O Points	I/O Range
8	0 to 255
16	0 to 65535



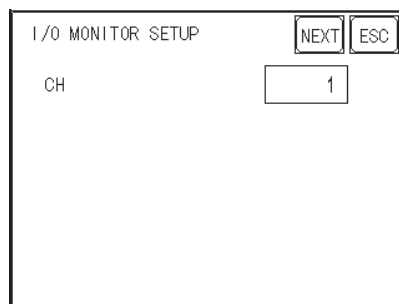
- **Data will be output to the I/O unit for the number of I/O points according to the MODEL selected on the I/O MONITOR SETUP window.**
- **If data that cannot be expressed in the 8-bit system is entered in an 8-point output I/O unit, excess data will be ignored.**



◆ **For FN-AD04AH/FN-DA04AH**

**I/O MONITOR (Channel Setting)**

The system switches successively through the selectable settings when the channel area is pressed.

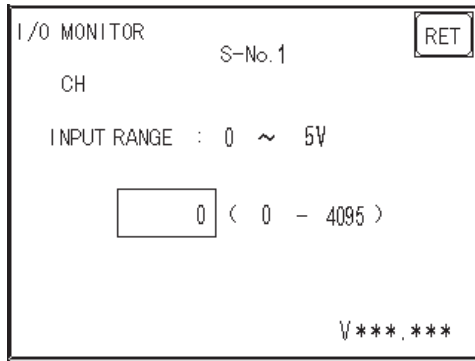


When the NEXT button is pressed, the system switches to the next I/O MONITOR screen. The screen is different for FN-AD04AH and FN-DA04AH.

### ◆ For FN-AD04AH

#### I/O MONITOR

This displays input data.



Pressing the [RET(URN)] button returns control to the [I/O MONITOR] screen.

#### A/D Conversion Table

Input Range Setting	Input Range
0 ~ 5V	0 ~ 4095
1 ~ 5V	0 ~ 4095
0 ~ 10V	0 ~ 4095
-5 ~ 5V	-2048 ~ 2047
-10 ~ 10V	-2048 ~ 2047
0 ~ 20mA	0 ~ 4095
4 ~ 20mA	0 ~ 4095



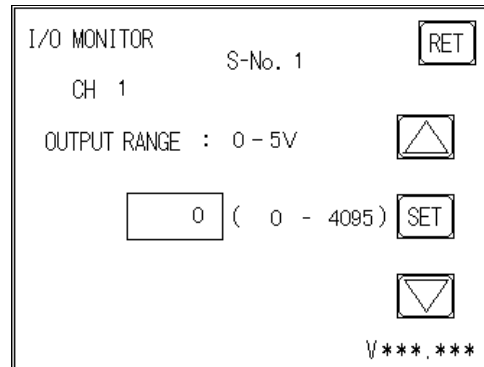
- **The filter type, A/D conversion sampling counts and maximum/minimum elimination settings will operate under the settings saved in the I/O unit. To change the settings saved in the I/O unit, change the settings via the LT Editor first, then download the ladder logic program to the LT. The changed settings will become effective when the ladder logic program is set to "RUN" mode.**
- **The setting for the Range Selector switch is loaded into the unit only upon power-up of the I/O unit. When changing the setting for the Range Selector switch, make sure to turn off the power to the I/O unit and then turn on the power again.**
- **The settings of the range changeover switch in the I/O unit side are read in when the logic program is switched to RUN mode. To change the settings of the range changeover switch, switch the logic program to STOP mode and then to RUN mode. If the ranges do not match, the data cannot be read correctly.**



## ◆ For FN-DA04AH

**I/O MONITOR**

Enter data with the keypad. With the LT unit, touching the screen's data display will call up the keypad. After entering all data, push the [OUT] button to output the data. All data is displayed in decimal format.



- **Touch the up and down arrow to increase/decrease the range value. Each time the value is changed, the new value is output to the I/O unit.**
- **Pressing the [RET(URN)] button will clear the current data, even if the output hold setting in the I/O unit is ON.**

**D/A Conversion Table**

Input Range Setting	Input Range
0 ~ 5V	0 ~ 4095
1 ~ 5V	0 ~ 4095
0 ~ 10V	0 ~ 4095
-5 ~ 5V	-2048 ~ 2047
-10 ~ 10V	-2048 ~ 2047
0 ~ 20mA	0 ~ 4095
4 ~ 20mA	0 ~ 4095



- **The setting for the Range Selector switch is loaded into the unit only when the I/O unit's power is turned on. When changing the Range Selector switch's setting, make sure to turn the I/O unit's power off and then on.**
- **The I/O unit's range changeover switch settings are read in when the logic program is switched to RUN mode. To change these settings, switch the logic program to STOP mode and then to RUN mode. If the ranges do not match, the data cannot be read correctly.**

## 11.2.5 Flex Network I/F Unit Troubleshooting

The following is a description of possible problems that may occur when using the Flex Network I/F unit, and their solutions.

### ■ Flex Network I/F unit I/O Errors

**Reference** For a detailed explanation of Flex Network I/F unit I/O errors, refer to the appropriate *Flex Network Unit Users Manual*.

### ■ Error Codes

I/O errors include those occurring during writing and reading. When one of these errors occurs, the controller writes an error code to #IOStatus.

### ◆ Setting Errors

Error Code	Problem	Solution
501	Internal variable error mapped to I/O terminal.	Reset the variable used.
502	External variable error mapped to I/O terminal.	
503	Output variable error mapped to I/O terminal.	
504	Discrete variable error mapped to analog terminal.	
505	Integer variable error mapped to discrete terminal.	
506	Variable type not supported by driver.	Correct the variable type.
507	Variable is not mapped to terminal.	Map the variable to all terminals.
801	Terminal numbers are duplicated.	The LTE file may be damaged or a malfunction has occurred during downloading of the lte file.
802	Multiple S-No. exist.	Two or more areas are using the same area number, possibly causing transfer failure. Download the project file again.
803	S-No. is outside of accepted range.	The LTE file may be damaged or a malfunction has occurred during downloading of the lte file.
804	S-No. range overlap at the analog unit.	Two or more I/O units are using the same S-No. The analog unit uses S-Nos. for four stations. Reset so there is no S-No. overlap.
805	S-No. range overlap with high-speed counter unit.	Two or more I/O units are using the same S-No. The high-speed counter unit has S-Nos. for eight stations. Reset so there is no S-No. overlap.
806	S-No. range overlap with single-axis positioning unit.	Two or more I/O units are using the same S-No. The positioning unit uses S-Nos. for four stations. Reset so there is no S-No. overlap.

## ◆ Initialization Errors

Error Code	Problem	Solution
821	There is no Flex Network I/F unit attached.	The ID Number loaded from the LT unit's built-in Flex Network I/F is invalid. The Flex Network I/F unit may be broken. Write down the error code and contact your local Pro-face distributor.
822	Initialization Error. Initialization failed to synchronize the Flex Network I/F unit and the Flex Network driver.	The Flex Network I/F unit may be broken. Write down the error code and contact your local Pro-face distributor.
823	Analog unit setting error	There may be a break in the communication cable, the I/O unit is not turned on, or the I/O unit may be broken.

## ◆ Runtime Errors

Error Code	Problem	Solution
841	There is an I/O unit error (loose connector, malfunction, etc.)	Check all related wiring. ▼ <b>Reference</b> Refer to the <i>Flex Network User Manual</i> (sold separately).
842	Disconnected output signal line of analog unit input sensor (A/D conversion unit)	This is likely due to disconnection in the output signal line. Check the sensor's output signal line.
843	Error in the high-speed counter unit	The High-Speed Counter unit detected an error. ▼ <b>Reference</b> Refer to the <i>Flex Network High-Speed Counter User Manual</i> (sold separately).
844	Initialization error in the high-speed counter unit	Check to see if the communication line is disconnected, power is not supplied to the I/O unit, or the I/O unit is malfunctioning.
845	Communication error with the high-speed counter unit	Check to see if the communication line is disconnected, power is not supplied to the I/O unit, or the I/O unit is malfunctioning.
846	Error in the single-axis positioning unit	The positioning unit detected an error. ▼ <b>Reference</b> Refer to the <i>Flex Network Single-Axis Positioning Unit User Manual</i> (sold separately).
847	Communication error with the single-axis positioning unit.	Check to see if communication line is disconnected, power is not supplied to the I/O unit, or the I/O unit is malfunctioning.

## ◆ Internal Errors

Error Code	Contents	Solution
850	Driver Error. A major system error has occurred.	Reset the LT. If an error code still appears, try to identify if the error is due to the LT itself, or to a related/connected device.
...		▼ <b>Reference</b> Write down the error code and refer to the <i>LT User Manual</i> .
859		Contact your local Pro-face distributor.

## 11.3 DIO Driver

This section explains the LT OFFLINE mode's DIO menu. The DIO driver should be downloaded from the LT Editor, before executing the DIO menu. The DIO driver is used with the LT Type A.

**Reference** For instructions on how to move to the OFFLINE mode screen, refer to the *LT Series User Manual* (sold separately).

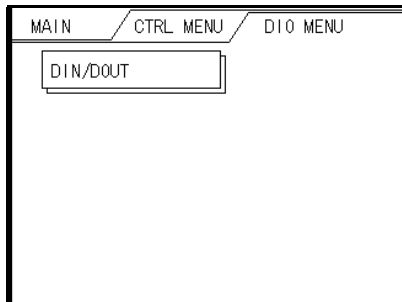
### 11.3.1 DIO Unit Self-Diagnosis

This section explains how to use the DIO unit's Self-Diagnosis feature.

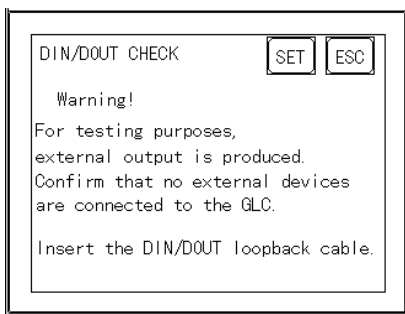
**Reference** For detailed information, refer to the *LT Series User Manual* (sold separately).

Touch the OFFLINE screen's Controller Menu to open the DIO Menu area.

<When DIO Driver has been Selected>



Touch the DIN/DOUT key to open the following screen.

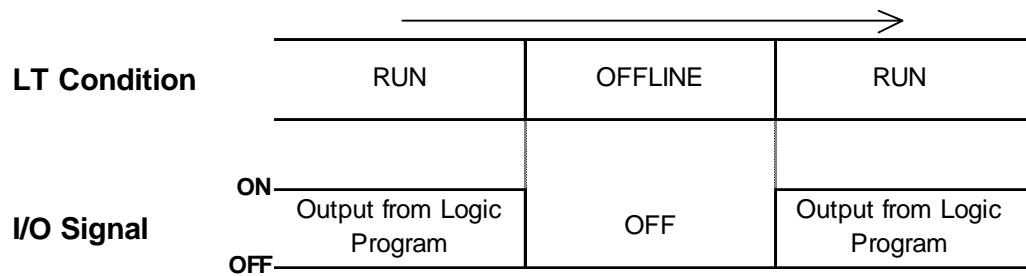


Touch either the [Set] or [Start] key to start the self-diagnosis.

This check sends an output signal from the output unit to the input unit. Therefore, prior to performing this check, be sure to attach the DIN/DOUT loopback cable.

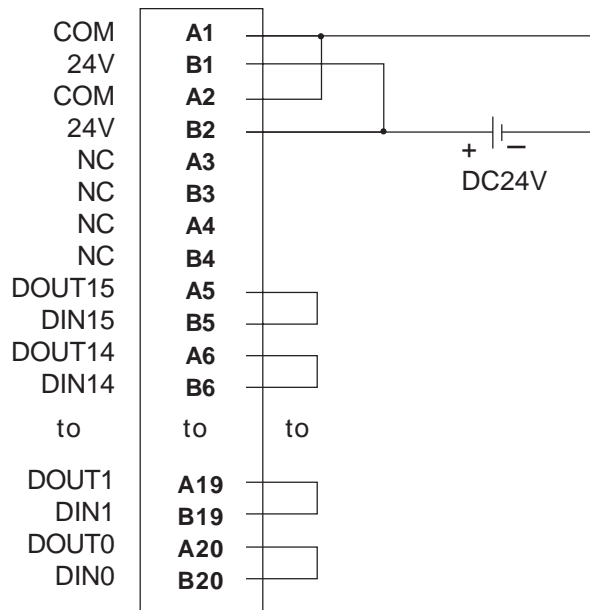


**When switching to the OFFLINE mode or when resetting from the logic program's RUN state, the I/O signal may turn OFF. Be aware of the possibility that the I/O signal will turn OFF.**



**The RESET mode's I/O signal OFF timing is NOT fixed.**

Use the following diagram when creating the DIN/DOUT loopback cable.



**Recommended Products**

Connection Type	Manufacturer	Model Number
Soldered Type	Fujitsu	FCN-361J040-AU (Connector) FCN-360C040-B (Cover)
Crimped Type	Fujitsu	FCN-363J040 FCN-363J-AU/S FCN-360C040-B
Terminal Block Unit Type	Mitsubishi	A6TBX36 (Terminal Block) AC**TB (Cable) (** = cable length)
	Yokogawa	TA40-ON

**11.3.2 I/O Monitor (I/O Connection Check)**

On the DIO driver menu touch [I/O MONITOR] to call up the following screen:

**<When I/O Monitor has been Selected>**

I/O MONITOR SETTINGS    NEXT    ESC

MODULE NUMBER (No.0-1)    0

INPUT TERMINALS    DISCRETE

OUTPUT TERMINALS    WORD

Select the [INPUT TERMINALS], either [DISCRETE] or [WORD].

Select the [OUTPUT TERMINALS], either [Discrete] or [Word].

For example, if you entered an [INPUT TERMINALS] of [DISCRETE] and an [OUTPUT TERMINALS] of [Word], and touched the screen’s upper right corner “RUN” button, the [I/O MONITOR] screen would appear.

I/O MONITOR    MODULE No.    RET

INPUT

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15

OUTPUT

      (0-65535)    OUT

When the [INPUT TERMINALS] is [DISCRETE], the input terminal (S-No.) will appear in reverse color. When the [OUTPUT TERMINALS] is [WORD], use the ten-key keypad to enter the data. When using a LT unit, touch the data entry field and the ten-key keypad will appear. After entering data, touch the [OUT] key to output the data. Data will be displayed in the decimal format.

### 11.3.3 DIO Unit Troubleshooting

This area explains how to solve possible DIO unit problems.

#### ■ DIO Unit Input Errors

Error Type	Possible Cause	Solution
Input monitor lamp is ON, but no input can be performed.	DIO Unit is defective.	Contact your local Pro-face distributor.
	[Enable I/O] box is not selected.	Set the [Enable I/O].
	Program is incorrect.	Correct program
Input monitor lamp is OFF and no input can be performed.	DIO Unit is defective	Contact your local Pro-face distributor.
	Input common line is incorrectly wired.	Common line wiring check. Common line breakage check. Common terminal looseness check.
	External input power is incorrect.	Provide the correct voltage.
	Connector is not securely attached.	Attach the connector securely.
All input lines do not turn OFF	DIO Unit is defective.	Contact your local Pro-face distributor.
Designated Input lines do not turn ON.	DIO Unit is defective.	Contact your local Pro-face distributor.
	Program is incorrect	Correct the program.
	Input wiring is incorrect.	Check common line wiring. Check common line breakage. Check common terminal for looseness.
	External unit is defective.	Replace the unit.
	Input ON period is too short.	Lengthen the Input ON time.
Designated Input lines do not turn OFF.	DIO Unit is defective.	Contact your local Pro-face distributor.
	Program is incorrect	Correct the program.
Input area randomly turns ON or OFF.	External Input voltage is incorrect.	Provide the correct voltage.
	Input terminal screws are loose.	Tighten the terminal screws.
	Program is incorrect	Correct the program.
	Connector is not securely attached.	Attach the connector securely.
	Noise is causing unit mis-operation.	Reduce the noise level. Attach a surge killer. Use a shielded cable.

■ DIO Unit Output Errors

Error Type	Possible Cause	Solution
Output monitor lamp is ON, but no output can be performed	DIO unit is defective.	Contact your local Pro-face distributor.
	Output common line is incorrectly wired.	Output line wiring check. Output line breakage check. Output terminal looseness check.
	Load current is incorrect.	Provide the correct current.
	Connector is not securely attached.	Attach the connector securely.
Output monitor lamp is OFF and no output can be performed	DIO unit is defective.	Contact your local Pro-face distributor.
	Program is incorrect Output area is completely OFF.	Correct program.
	[Enable I/O] box is not selected.	Set the [Enable I/O].
Output lines do not turn OFF	DIO unit is defective.	Contact your local Pro-face distributor.
Designated output lines do not turn ON	DIO unit is defective.	Contact your local Pro-face distributor.
	Output wiring is incorrect	Check output line wiring. Check output line breakage. Check output terminal for looseness.
	External unit is defective.	Replace unit.
Designated output lines do not go OFF	DIO unit is defective.	Contact your local Pro-face distributor.
	Current leakage, residual voltage causes incorrect recurrence.	Change design of external device. I.e. Attach dummy resistor, etc.
Output area randomly turns ON/OFF	Load voltage is incorrect.	Correct voltage load.
	Output terminal screws are loose.	Tighten the terminal screws.
	Program is incorrect. Output commands are overlapping.	Correct the program.
	Connector is not securely attached.	Attach the connector securely.
	Noise is causing unit operation error.	Reduce the noise level. Attach a surge killer. Use a shielded cable.



## ■ Error Codes

I/O errors are Read/Write errors. When I/O errors occur, the controller writes an error code to the #IOStatus variable. The logic program continues to operate. The following explanation of possible error causes and solutions for when the DIO unit is attached to the LT.

### ◆ Setting Errors

Error Code	Contents	Solution
501	Internal variable error allocated to I/O terminal.	Reset the variable used.
502	External variable error allocated to I/O terminal.	
503	Output variable error allocated to I/O terminal.	
504	Discrete variable error allocated to analog terminal.	
505	Integer variable error allocated to discrete terminal.	
506	Variable type not supported by driver.	Correct the variable type.
801	Terminal numbers are duplicated.	Two or more terminals are using the same terminal number, possibly causing data transfer failure. Download the LTE file again.
802	Multiple modules are used.	Two DIO units are using the same module number. Reset these numbers so they do not overlap.
803	Module number has exceeded 1.	Set a module number from 0 to 1.
804	Unit number starts from 1.	Set the DIO unit nearest the rear face of the LT to "0".
805	Driver configuration error	More than one DIO driver has been added to the I/O tree.
821	No DIO hardware unit exists.	LTE file contains more modules than the actual number of connected LTs
822	DIO-: No hardware unit exists, or the type is invalid.	The ID Number loaded from the built-in DIO unit is invalid. The DIO unit may be broken.

### ◆ Runtime Errors

Error Code	Contents	Solution
840	Read-out data is incorrect. After two successive read attempts, the LT has detected that the value read out from the DIO is incorrect.	Increase the time of the Input signal's ON period.
841	Output data is incorrect. Incorrect output data was detected from the built-in DIO unit by an internal loopback check.	Check to see if there is noise interference.
842	DIO output data is incorrect	Incorrect output data was detected by an internal loopback check.

### ◆ Internal Errors

Error Code	Contents	Solution
850 ... 864	Driver Error A major system error has occurred.	Record the Error Number and contact your local Pro-face distributor.

# 12 Error Messages

## 12.1 Error Message List

This chapter explains the error messages that can appear on the LT unit. The error messages explained here are those related to the LT Editor program only.

For further information concerning LT error messages, refer to

**Reference** *LT Series User Manual (Sold separately)*

Error Message	Cause	Solution
"Invalid ladder file"	The LT's logic program file is not downloaded, or the file is damaged.	Download another copy of the project file from the LT Editor.
"Fatal Error: Drive check Failed"	The LT's current I/O driver is incorrect.	Check that the I/O driver set in the logic program file and the driver installed in the LT are the same.
"Global Data Area Too Small"	The downloaded file's data may be damaged.	Download the project file again. If this does not fix the problem, contact your local Pro-face dealer.
"Can't Set Priority"	The LT's system file is incorrect. The file may have been damaged during downloading.	Download the project file again.
"Exception nnn:[mmm:ooo]"	A fatal error has occurred in the ladder logic program.	Write down the error message details and consult your local Pro-face dealer.
"Watchdog Error"	The Constant Scan Time is longer than the Watchdog time.	Reset the Watchdog time so that it is longer than the Constant Scan Time. If doing so exceeds the Watchdog Timer's limit, then the Constant Scan Time (program) should be changed.

## Chapter 12 – Error Messages

Error Message	Cause	Solution
"Bad Var: xxx"	Unable to find variable "XXX". Either the logic program file has not been downloaded, or a variable that does not exist in the logic program file on the screen, is used.	Download the project file again.
"Bad Array: xxx"	The number of elements used in the screen file's array variables and those used in the logic program file's array variables are different.	Download the project file again.
"Bad T ype xxx"	The Logic program variable "XXX"'s type is different from the	Download the project file again.
"Unknown register type"	This variable type does not	Download the project file again.
"Register is missing"	Cannot find variable used for	
"S100 file index is out of range"	Cannot find variable used for Reading.	
"Too many entries in the S100 file"	Too many variables are being used. Limit is 2048.	
"S100 file is missing"	Cannot find S100 (variable	
"Over Compile count MAX"	Too many Parts are being used.	Reduce the number of Parts and then download the project to the LT again.
"Exception 65532 [xxxx : xxx] " "Exception 65533 [xxxx : xxx] " "Exception 65534 [xxxx : xxx] " "Exception 65535 [xxxx : xxx] "	LT heap memory is insufficient. Memory for storing programs and variables is sufficient, however logic program memory is insufficient.	Setup the LT unit again with the LT Editor after reducing the logic program size, or the number of variables and labels. Also reduce the number of array variable elements, or shorten the name of variables and labels.

## 12.2 Error Codes

The following table describes #FaultCode errors.

Error Code	Level	Cause
0	Normal	No error
1	Minor	The calculation result, or the result of the conversion of a Real variable to an Integer variable has resulted in
2	Major	A reference was used for an area outside the array's
3	Major	A reference was used for a bit outside the Integer's (32 bit) range
4	Major	The stack has overflowed.
5	Major	Incorrect command code is being used.
6	-	Reserved for System.
7	Major	The Scan time is now longer than the Watchdog time.
8	-	Reserved for System.
9	Major	Software Error. Depending on type of problem, system may need to be restarted.
10	-	Reserved for System.
11	-	Reserved for System.
12	Minor	BCD/BIN Conversion Error
13	Minor	BCD/BIN Conversion Error
14	-	Reserved for System.



### Major Faults and Minor Faults

- **When a major error occurs, the controller immediately stops executing the logic program.**
- **When a minor error occurs, the controller is able to continue executing the logic program.**

## 12.3 Program Errors

The following table explains the LT Editor's logic program operation errors.

Error Type	Possible Problem	Solution
Control Memory power is cut and Hold Area data is not preserved	Battery Alarm	Change Battery
	Memory Alarm	Exchange Unit
Program Malfunction	Program transfer mistake.	Download the project file again with LT Editor.
Data is output from I/O even in STOP mode.	When output data performs RUN/STOP switchover, I/O output hold is enabled.	Disable this feature.
Soon after entering RUN mode unit changes to STOP mode	A Command Execution Alarm has occurred. Or, a major fault has occurred.	Modify the program. Check the contents of the #FaultCode data. Also check if the System variable #Command has been written to.
LT Editor cannot enter Monitoring mode	The data transfer cable used to send data from the screen creation software to the LT unit may be loose or disconnected. Also, the PC or LT's power may have dropped, causing excessive noise.	Check whether the data transfer cable is unplugged or if there is noise interference. If the problem continues, please contact your local Pro-face distributor for assistance.
The logic program files cannot be downloaded from LT Editor		
The project (.prw) file cannot be downloaded from the LT Editor.		
Data cannot be read from or written to the I/O.	Enable I/O*1 is not selected.	Set the Enable I/O.

1. *Enable I/O is used to input and output data between the LT and I/O units. After downloading the logic program to the LT unit, the external I/O devices cannot operate in RUN mode. (As a safety precaution, the I/O is not enabled in the default setting.) Be sure to set the Enable I/O before trying to read/write data to I/O.*

**Reference** For setup details, refer to the **Programming Guide**.