OMRON

Industrial PC Platform

NY-series

Instructions Reference Manual

NY532-1500
NY532-1400
NY532-1300
NY532-5400
NY512-1500
NY512-1400
NY512-1300

SYSMAC
always in control

W560-E1-06

NOTE

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, mechanical, electronic, photocopying, recording, or otherwise, without the prior written permission of OMRON.

No patent liability is assumed with respect to the use of the information contained herein. Moreover, because OMRON is constantly striving to improve its high-quality products, the information contained in this manual is subject to change without notice. Every precaution has been taken in the preparation of this manual. Nevertheless, OMRON assumes no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained in this publication.

Trademarks

- Sysmac and SYSMAC are trademarks or registered trademarks of OMRON Corporation in Japan and other countries for OMRON factory automation products.
- Microsoft, Windows, Excel, and Visual Basic are either registered trademarks or trademarks of Microsoft Corporation in the United States and other countries.
- EtherCAT® is registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany.
- ODVA, CIP, CompoNet, DeviceNet, and EtherNet/IP are trademarks of ODVA.
- The SD and SDHC logos are trademarks of SD-3C, LLC.
- Intel and Intel Core are trademarks of Intel Corporation in the U.S. and / or other countries.

Other company names and product names in this document are the trademarks or registered trademarks of their respective companies.

Copyrights

Microsoft product screen shots reprinted with permission from Microsoft Corporation.

Introduction

Thank you for purchasing an NY-series IPC Machine Controller Industrial Panel PC / Industrial Box PC. This manual provides a collective term of Industrial Panel PC and Industrial Box PC which are applicable products as the NY-series Industrial PC. This manual also provides the range of devices that are directly controlled by the Controller functions embedded the Real-Time OS in the NY-series Industrial PC as the Controller.

This manual contains information that is necessary to use the NY-series Controller. Please read this manual and make sure you understand the functionality and performance of the NY-series Controller before you attempt to use it in a control system.
Keep this manual in a safe place where it will be available for reference during operation.

Intended Audience

This manual is intended for the following personnel, who must also have knowledge of electrical systems (an electrical engineer or the equivalent).

- Personnel in charge of introducing FA systems.
- Personnel in charge of designing FA systems.
- Personnel in charge of installing and maintaining FA systems.
- Personnel in charge of managing FA systems and facilities.

For programming, this manual is intended for personnel who understand the programming language specifications in international standard IEC 61131-3 or Japanese standard JIS B 3503.

Applicable Products

This manual covers the following products.

- NY-series IPC Machine Controller Industrial Panel PC
  - NY532-15□□
  - NY532-14□□
  - NY532-13□□
  - NY532-5400

- NY-series IPC Machine Controller Industrial Box PC
  - NY512-15□□
  - NY512-14□□
  - NY512-13□□

Part of the specifications and restrictions for the Industrial PC are given in other manuals. Refer to *Relevant Manuals* on page 2 and *Related Manuals* on page 24.

Relevant Manuals

The following table provides the relevant manuals for the NY-series Controller.

Read all of the manuals that are relevant to your system configuration and application before you use the NY-series Controller.

Most operations are performed from the Sysmac Studio Automation Software. Refer to the *Sysmac Studio Version 1 Operation Manual* (Cat. No. W504) for information on the Sysmac Studio.

Purpose of use	Manual										
	Basic information										
	NY-series IPC Machine Controller Industrial Panel PC Hardware User's Manual	NY-series IPC Machine Controller Industrial Box PC Hardware User's Manual	NY-series IPC Machine Controller Industrial Panel PC / Industrial Box PC Setup User's Manual	NY-series IPC Machine Controller Industrial Panel PC / Industrial Box PC Software User's Manual	NY-series Instructions Reference Manual	NY-series IPC Machine Controller Industrial Panel PC / Industrial Box PC Motion Control User's Manual	NY-series Motion Control Instructions Reference Manual	NY-series IPC Machine Controller Industrial Panel PC / Industrial Box PC Built-in EtherCAT Port User's Manual	NY-series IPC Machine Controller Industrial Panel PC / Industrial Box PC Built-in EtherNet/IP Port User's Manual	NJ/NY-series NC Integrated Controller User's Manual	NY-series Troubleshooting Manual
Introduction to NY-series Panel PCs	○										
Introduction to NY-series Box PCs		○									
Setting devices and hardware	○	○									
Using motion control						○					
Using EtherCAT								○			
Using EtherNet/IP									○		
Making setup*1			○								
Making initial settings											
Preparing to use Controllers											
Software settings				○							
Using motion control						○					
Using EtherCAT								○			
Using EtherNet/IP									○		
Using numerical control										○	
Writing the user program				○	○						
Using motion control						○	○				
Using EtherCAT								○			
Using EtherNet/IP									○		
Using numerical control										○	
Programming error processing											○
Testing operation and debugging				○							
Using motion control						○					
Using EtherCAT								○			
Using EtherNet/IP									○		
Using numerical control										○	
Learning about error management and corrections*2											○
Maintenance	○	○									
Using motion control						○					
Using EtherCAT								○			
Using EtherNet/IP									○		

*1 Refer to the *NY-series Industrial Panel PC / Industrial Box PC Setup User's Manual* (Cat. No. W568) for how to set up and how to use the utilities on Windows.

*2 Refer to the *NY-series Troubleshooting Manual* (Cat. No. W564) for the error management concepts and the error items.

Manual Structure

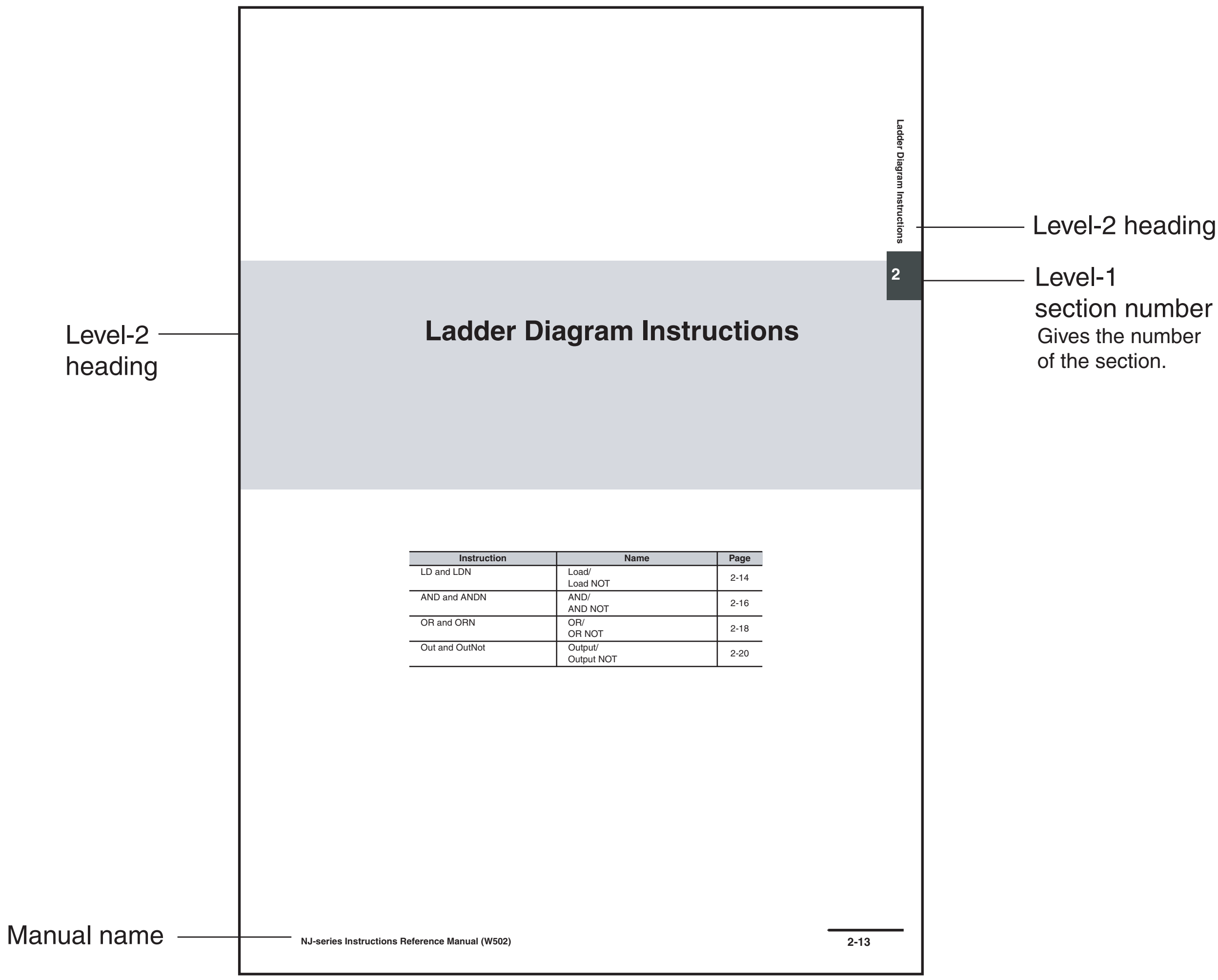
Some of the instructions described in this manual are common to the NJ/NX-series.

Therefore, note the following conditions.

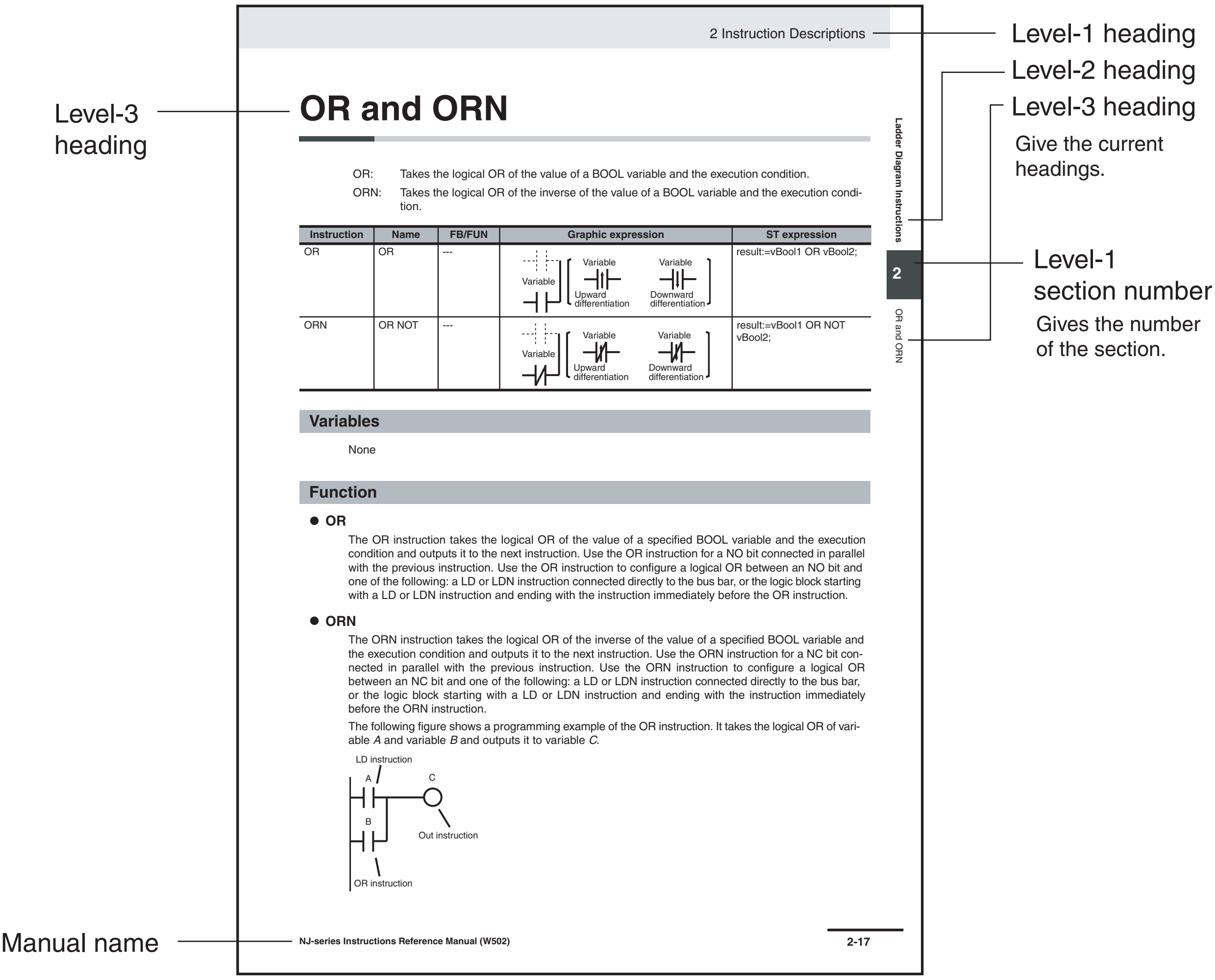
- You cannot connect a CJ-series Unit with NY-series Controllers. In the instructions, skip items and samples related to CJ-series Units.
- In explanation of the instructions, replace the term “CPU Unit” with “NY-series Controller.”
- NY-series Controllers have no SD Memory Card slots. Instead, they provide the Virtual SD Memory Card function that uses the Windows shared folder. Therefore, replace the term “SD Memory Card” with “Virtual SD Memory Card.” Refer to the *NY-series Industrial Panel PC / Industrial Box PC Software User’s Manual* (Cat. No. W558) and *NY-series Industrial Panel PC / Industrial Box PC Setup User’s Manual* (Cat. No. W568) for details on the function of a Virtual SD Memory Card.
- The unit version of the NY-series Controller is 1.12 or later.

Page Structure

The following page structure is used in this manual.



This page is for illustration only. It may not literally appear in this manual.



2 Instruction Descriptions

Ladder Diagram Instructions

2

OR and ORN

OR and ORN

OR: Takes the logical OR of the value of a BOOL variable and the execution condition.

ORN: Takes the logical OR of the inverse of the value of a BOOL variable and the execution condition.

Instruction	Name	FB/FUN	Graphic expression	ST expression
OR	OR	---	Variable; Variable Upward differentiation; Variable Downward differentiation	result:=vBool1 OR vBool2;
ORN	OR NOT	---	Variable; Variable Upward differentiation; Variable Downward differentiation	result:=vBool1 OR NOT vBool2;

Variables

None

Function

● **OR**

The OR instruction takes the logical OR of the value of a specified BOOL variable and the execution condition and outputs it to the next instruction. Use the OR instruction for a NO bit connected in parallel with the previous instruction. Use the OR instruction to configure a logical OR between an NO bit and one of the following: a LD or LDN instruction connected directly to the bus bar, or the logic block starting with a LD or LDN instruction and ending with the instruction immediately before the OR instruction.

● **ORN**

The ORN instruction takes the logical OR of the inverse of the value of a specified BOOL variable and the execution condition and outputs it to the next instruction. Use the ORN instruction for a NC bit connected in parallel with the previous instruction. Use the ORN instruction to configure a logical OR between an NC bit and one of the following: a LD or LDN instruction connected directly to the bus bar, or the logic block starting with a LD or LDN instruction and ending with the instruction immediately before the ORN instruction.

The following figure shows a programming example of the OR instruction. It takes the logical OR of variable *A* and variable *B* and outputs it to variable *C*.

LD instruction
A
C
B
Out instruction
OR instruction

NJ-series Instructions Reference Manual (W502) 2-17

This page is for illustration only. It may not literally appear in this manual.

Version Information

Information on differences in specifications and functionality for CPU Units with different unit versions and for different versions of the Sysmac Studio are given.

Sections in this Manual

1
2
A
I

CONTENTS

Appendices

Index

Terms and Conditions Agreement

Warranty, Limitations of Liability

Warranties

- **Exclusive Warranty**

  Omron's exclusive warranty is that the Products will be free from defects in materials and workmanship for a period of twelve months from the date of sale by Omron (or such other period expressed in writing by Omron). Omron disclaims all other warranties, express or implied.

- **Limitations**

  OMRON MAKES NO WARRANTY OR REPRESENTATION, EXPRESS OR IMPLIED, ABOUT NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE OF THE PRODUCTS. BUYER ACKNOWLEDGES THAT IT ALONE HAS DETERMINED THAT THE PRODUCTS WILL SUITABLY MEET THE REQUIREMENTS OF THEIR INTENDED USE.

  Omron further disclaims all warranties and responsibility of any type for claims or expenses based on infringement by the Products or otherwise of any intellectual property right.

- **Buyer Remedy**

  Omron's sole obligation hereunder shall be, at Omron's election, to (i) replace (in the form originally shipped with Buyer responsible for labor charges for removal or replacement thereof) the non-complying Product, (ii) repair the non-complying Product, or (iii) repay or credit Buyer an amount equal to the purchase price of the non-complying Product; provided that in no event shall Omron be responsible for warranty, repair, indemnity or any other claims or expenses regarding the Products unless Omron's analysis confirms that the Products were properly handled, stored, installed and maintained and not subject to contamination, abuse, misuse or inappropriate modification. Return of any Products by Buyer must be approved in writing by Omron before shipment. Omron Companies shall not be liable for the suitability or unsuitability or the results from the use of Products in combination with any electrical or electronic components, circuits, system assemblies or any other materials or substances or environments. Any advice, recommendations or information given orally or in writing, are not to be construed as an amendment or addition to the above warranty.

See http://www.omron.com/global/ or contact your Omron representative for published information.

Limitation on Liability; Etc

OMRON COMPANIES SHALL NOT BE LIABLE FOR SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, LOSS OF PROFITS OR PRODUCTION OR COMMERCIAL LOSS IN ANY WAY CONNECTED WITH THE PRODUCTS, WHETHER SUCH CLAIM IS BASED IN CONTRACT, WARRANTY, NEGLIGENCE OR STRICT LIABILITY.

Further, in no event shall liability of Omron Companies exceed the individual price of the Product on which liability is asserted.

Application Considerations

Suitability of Use

Omron Companies shall not be responsible for conformity with any standards, codes or regulations which apply to the combination of the Product in the Buyer's application or use of the Product. At Buyer's request, Omron will provide applicable third party certification documents identifying ratings and limitations of use which apply to the Product. This information by itself is not sufficient for a complete determination of the suitability of the Product in combination with the end product, machine, system, or other application or use. Buyer shall be solely responsible for determining appropriateness of the particular Product with respect to Buyer's application, product or system. Buyer shall take application responsibility in all cases.

NEVER USE THE PRODUCT FOR AN APPLICATION INVOLVING SERIOUS RISK TO LIFE OR PROPERTY OR IN LARGE QUANTITIES WITHOUT ENSURING THAT THE SYSTEM AS A WHOLE HAS BEEN DESIGNED TO ADDRESS THE RISKS, AND THAT THE OMRON PRODUCT(S) IS PROPERLY RATED AND INSTALLED FOR THE INTENDED USE WITHIN THE OVERALL EQUIPMENT OR SYSTEM.

Programmable Products

Omron Companies shall not be responsible for the user's programming of a programmable Product, or any consequence thereof.

Disclaimers

Performance Data

Data presented in Omron Company websites, catalogs and other materials is provided as a guide for the user in determining suitability and does not constitute a warranty. It may represent the result of Omron's test conditions, and the user must correlate it to actual application requirements. Actual performance is subject to the Omron's Warranty and Limitations of Liability.

Change in Specifications

Product specifications and accessories may be changed at any time based on improvements and other reasons. It is our practice to change part numbers when published ratings or features are changed, or when significant construction changes are made. However, some specifications of the Product may be changed without any notice. When in doubt, special part numbers may be assigned to fix or establish key specifications for your application. Please consult with your Omron's representative at any time to confirm actual specifications of purchased Product.

Errors and Omissions

Information presented by Omron Companies has been checked and is believed to be accurate; however, no responsibility is assumed for clerical, typographical or proofreading errors or omissions.

Safety Precautions

Refer to the following manuals for safety precautions.

- NY-series Industrial Box PC Hardware User’s Manual (Cat. No. W556)
- NY-series Industrial Panel PC Hardware User’s Manual (Cat. No. W557)
- NY-series Industrial Panel PC / Industrial Box PC Software User’s Manual (Cat. No. W558)

Precautions for Safe Use

Refer to the following manuals for precautions for safe use.

- NY-series Industrial Box PC Hardware User's Manual (Cat. No. W556)
- NY-series Industrial Panel PC Hardware User's Manual (Cat. No. W557)
- NY-series Industrial Panel PC / Industrial Box PC Software User's Manual (Cat. No. W558)

Precautions for Correct Use

Refer to the following manuals for precautions for correct use.

- NY-series Industrial Box PC Hardware User's Manual (Cat. No. W556)
- NY-series Industrial Panel PC Hardware User's Manual (Cat. No. W557)
- NY-series Industrial Panel PC / Industrial Box PC Software User's Manual (Cat. No. W558)

Regulations and Standards

Conformance to EU Directives

Applicable Directives

- EMC Directives

Concepts

● EMC Directive

OMRON devices that comply with EU Directives also conform to the related EMC standards so that they can be more easily built into other devices or the overall machine. The actual products have been checked for conformity to EMC standards.*
Whether the products conform to the standards in the system used by the customer, however, must be checked by the customer. EMC-related performance of the OMRON devices that comply with EU Directives will vary depending on the configuration, wiring, and other conditions of the equipment or control panel on which the OMRON devices are installed. The customer must, therefore, perform the final check to confirm that devices and the overall machine conform to EMC standards.

* Applicable EMC (Electromagnetic Compatibility) standards are as follows:
EMS (Electromagnetic Susceptibility): EN 61131-2
EMI (Electromagnetic Interference): EN 61131-2 (Radiated emission: 10-m regulations)

● Conformance to EU Directives

The NY-series Controllers comply with EU Directives. To ensure that the machine or device in which the NY-series Controller is used complies with EU Directives, the Controller must be installed as follows:

- The NY-series Controller must be installed within a control panel.
- You must use the power supply in SELV specifications for the DC power supplies connected to DC Power Supply Units and I/O Units.
- NY-series Controllers that comply with EU Directives also conform to the Common Emission Standard (EN 61000-6-4). Radiated emission characteristics (10-m regulations) may vary depending on the configuration of the control panel used, other devices connected to the control panel, wiring, and other conditions.
  You must therefore confirm that the overall machine or equipment complies with EU Directives.

Software Licenses and Copyrights

This product incorporates certain third party software. The license and copyright information associated with this software is available at http://www.fa.omron.co.jp/nj_info_e/.

Versions

Hardware revisions and unit versions are used to manage the hardware and software in NY-series Controllers and EtherCAT slaves. The hardware revision or unit version is updated each time there is a change in hardware or software specifications. Even when two Units or EtherCAT slaves have the same model number, they will have functional or performance differences if they have different hardware revisions or unit versions.

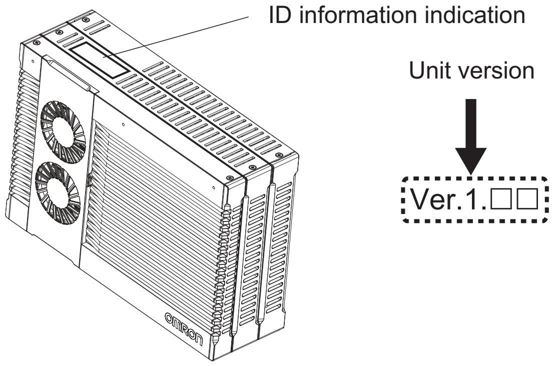
Checking Versions

You can check versions in the ID information indications or with the Sysmac Studio.

Checking Unit Versions on ID Information Indications

The unit version is given on the ID information indication on the back side of the product.

The ID information on an NY-series NY5□2-□□□□ Controller is shown below.



Checking Unit Versions with the Sysmac Studio

You can use the Sysmac Studio to check unit versions. The procedure is different for Units and for EtherCAT slaves.

● Checking the Unit Version of an NY-series Controller

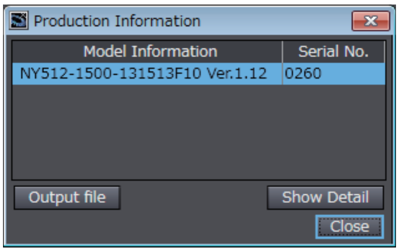
You can use the Production Information while the Sysmac Studio is online to check the unit version of a Unit. You can only do this for the Controller.

- Right-click **CPU Rack** under **Configurations and Setup** – **CPU/Expansion Racks** in the Multiview Explorer and select **Production Information**.
- The Production Information Dialog Box is displayed.

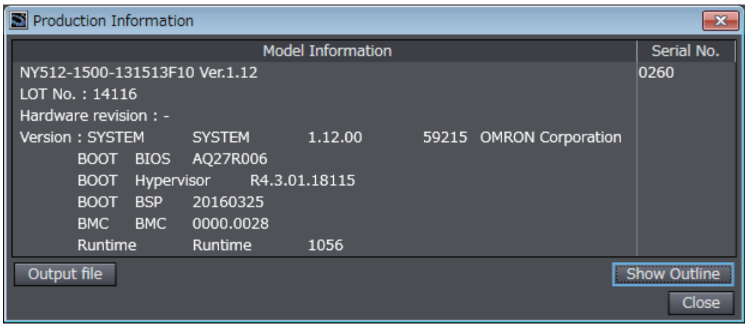
● Changing Information Displayed in Production Information Dialog Box

1 Click the **Show Detail** or **Show Outline** Button at the lower right of the Production Information Dialog Box.

The view will change between the production information details and outline.



Outline View



Detail View

The information that is displayed is different for the Outline View and Detail View. The Detail View displays the unit version, hardware revision, and other versions. The Outline View displays only the unit version.

● Checking the Unit Version of an EtherCAT Slave

You can use the Production Information while the Sysmac Studio is online to check the unit version of an EtherCAT slave. Use the following procedure to check the unit version.

1 Double-click **EtherCAT** under **Configurations and Setup** in the Multiview Explorer. Or, right-click **EtherCAT** under **Configurations and Setup** and select **Edit** from the menu.

The EtherCAT Tab Page is displayed.

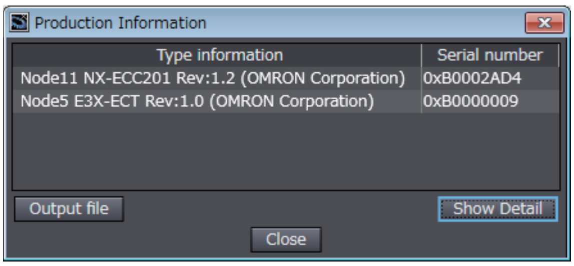
2 Right-click the master on the EtherCAT Tab Page and select **Display Production Information**.

The Production Information Dialog Box is displayed.
The unit version is displayed after “Rev.”

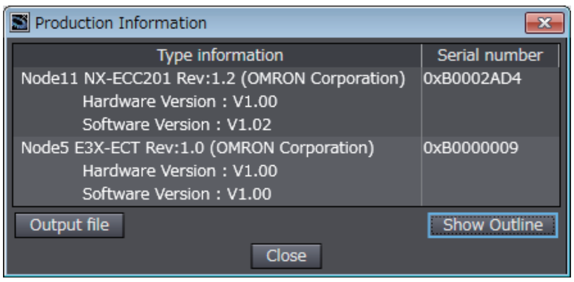
● Changing Information Displayed in Production Information Dialog Box

1 Click the **Show Detail** or **Show Outline** Button at the lower right of the Production Information Dialog Box.

The view will change between the production information details and outline.



Outline View



Detail View

Related Manuals

The followings are the manuals related to this manual. Use these manuals for reference.

Manual name	Cat. No.	Model numbers	Application	Description
NY-series IPC Machine Controller Industrial Panel PC Hardware User's Manual	W557	NY532-□□□□	Learning the basic specifications of the NY-series Industrial Panel PCs, including introductory information, designing, installation, and maintenance. Mainly hardware information is provided.	An introduction to the entire NY-series system is provided along with the following information on the Industrial Panel PC. • Features and system configuration • Introduction • Part names and functions • General specifications • Installation and wiring • Maintenance and inspection
NY-series IPC Machine Controller Industrial Box PC Hardware User's Manual	W556	NY512-□□□□	Learning the basic specifications of the NY-series Industrial Box PCs, including introductory information, designing, installation, and maintenance. Mainly hardware information is provided.	An introduction to the entire NY-series system is provided along with the following information on the Industrial Box PC. • Features and system configuration • Introduction • Part names and functions • General specifications • Installation and wiring • Maintenance and inspection
NY-series IPC Machine Controller Industrial Panel PC / Industrial Box PC Setup User's Manual	W568	NY532-□□□□ NY512-□□□□	Learning about initial setting of the NY-series Industrial PCs and how to prepare the Controller.	The following information is provided on an introduction to the entire NY-series system. • Two OS systems • Initial settings • Industrial PC Support Utility • NYCompolet • Industrial PC API • Backup and recovery
NY-series IPC Machine Controller Industrial Panel PC / Industrial Box PC Software User's Manual	W558	NY532-□□□□ NY512-□□□□	Learning how to program and set up the Controller functions of an NY-series Industrial PC.	The following information is provided on the NY-series Controller functions. • Controller operation • Controller features • Controller settings • Programming based on IEC 61131-3 language specifications
NY-series Instructions Reference Manual	W560	NY532-□□□□ NY512-□□□□	Learning detailed specifications on the basic instructions of the NY-series Industrial PC.	The instructions in the instruction set (IEC 61131-3 specifications) are described.
NY-series IPC Machine Controller Industrial Panel PC / Industrial Box PC Motion Control User's Manual	W559	NY532-□□□□ NY512-□□□□	Learning about motion control settings and programming concepts of an NY-series Industrial PC.	The settings and operation of the Controller and programming concepts for motion control are described.
NY-series Motion Control Instructions Reference Manual	W561	NY532-□□□□ NY512-□□□□	Learning about the specifications of the motion control instructions of an NY-series Industrial PC.	The motion control instructions are described.

Manual name	Cat. No.	Model numbers	Application	Description
NY-series IPC Machine Controller Industrial Panel PC / Industrial Box PC Built-in Eth-erCAT® Port User's Manual	W562	NY532-□□□□ NY512-□□□□	Using the built-in Eth-erCAT port in an NY-series Industrial PC.	Information on the built-in EtherCAT port is provided. This manual provides an introduction and provides information on the configuration, features, and setup.
NY-series IPC Machine Controller Industrial Panel PC / Industrial Box PC Built-in EtherNet/IP™ Port User's Manual	W563	NY532-□□□□ NY512-□□□□	Learning about the errors that may be detected in an NY-series Industrial PC.	Information on the built-in EtherNet/IP port is provided. Information is provided on the basic setup, tag data links, and other features.
NJ/NY-series NC Integrated Controller User's Manual	O030	NJ501-5300 NY532-5400	Performing numeri-cal control with NJ/NY-series Control-lers.	Describes the functionality to perform the numerical control.
NJ/NY-series G code Instruc-tions Reference Manual	O031	NJ501-5300 NY532-5400	Learning about the specifications of the G code/M code instructions.	The G code/M code instructions are described.
NY-series Troubleshooting Manual	W564	NY532-□□□□ NY512-□□□□	Learning about the errors that may be detected in an NY-series Industrial PC.	Concepts on managing errors that may be detected in an NY-series Controller and infor-mation on individual errors are described.
Sysmac Studio Version 1 Operation Manual	W504	SYSMAC -SE2□□□	Learning about the operating proce-dures and functions of the Sysmac Studio.	Describes the operating procedures of the Sysmac Studio.
CNC Operator Operation Manual	O032	SYSMAC -RTNC0□□□D	Learning an introduc-tion of the CNC Oper-ator and how to use it.	An introduction of the CNC Operator, installa-tion procedures, basic operations, connection operations, and operating procedures for main functions are described.
NX-series EtherCAT® Coupler Unit User's Manual	W519	NX-ECC□□□	Learning how to use an NX-series Ether-CAT Coupler Unit and EtherCAT Slave Ter-minals.	The following items are described: the overall system and configuration methods of an Eth-erCAT Slave Terminal (which consists of an NX-series EtherCAT Coupler Unit and NX Units), and information on hardware, setup, and functions to set up, control, and monitor NX Units through EtherCAT.
NX-series NX Units User's Manuals	W521	NX-ID□□□□ NX-IA□□□□ NX-OC□□□□ NX-OD□□□□	Learning how to use NX Units.	Describe the hardware, setup methods, and functions of the NX Units. Manuals are available for the following Units. Digital I/O Units, Analog I/O Units, System Units, and Position Interface Units.
	W522	NX-AD□□□□ NX-DA□□□□ NX-TS□□□□		
	W523	NX-PD1□□□ NX-PF0□□□ NX-PC0□□□ NX-TBX□□		
	W524	NX-EC0□□□ NX-ECS□□□ NX-PG0□□□		
NX-series Data Reference Manual	W525	NX-□□□□□□	Referring to the list of data required for NX-series unit system configuration.	Provides the list of data required for system configuration including the power consump-tion and weight of each NX-series Unit.

Manual name	Cat. No.	Model numbers	Application	Description
GX-series EtherCAT Slave Units User’s Manual	W488	GX-ID□□□□ GX-OD□□□□ GX-OC□□□□ GX-MD□□□□ GX-AD□□□□ GX-DA□□□□ GX-EC□□□□ XWT-ID□□ XWT-OD□□	Learning how to use the EtherCAT remote I/O terminals.	Describes the hardware, setup methods, and functions of the EtherCAT remote I/O terminals.
AC Servomotors/Servo Drives 1S-series with Built-in EtherCAT® Communications User’s Manual	I586	R88M-1□ R88D-1SN□-ECT	Learning how to use the Servomotors/Servo Drives with built-in EtherCAT Communications.	Describes the hardware, setup methods and functions of the Servomotors/Servo Drives with built-in EtherCAT Communications.
AC Servomotors/Servo Drives G5-series with Built-in EtherCAT® Communications User's Manual	I573	R88M-K□ R88D-KN□-ECT-R	Learning how to use the Servomotors/Servo Drives with built-in EtherCAT Communications.	Describes the hardware, setup methods and functions of the Servomotors/Servo Drives with built-in EtherCAT Communications. The linear motor type model and the model dedicated for position controls are available in G5-series.
	I576	R88M-K□ R88D-KN□-ECT		
	I577	R88L-EC-□ R88D-KN□-ECT-L		

Revision History

A manual revision code appears as a suffix to the catalog number on the front and back covers of the manual.

Revision code	Date	Revised content
01	September 2016	Original production
02	November 2016	Corrected mistakes.
03	April 2017	Corrected mistakes.
04	October 2017	• Made changes accompanying release of unit version 1.16 of the CPU Unit and version 1.20 of the Sysmac Studio. • Corrected mistakes.
05	April 2018	• Made changes accompanying release of unit version 1.18 of the CPU Unit and version 1.22 of the Sysmac Studio. • Corrected mistakes.
06	January 2019	• Corrected mistakes.

1

Instruction Set

This section provides a table of the instructions that you can use with NY-series Controllers.

Instruction Set

Type	Instruction	Name	Function	Page
Ladder Diagram Instructions	LD	Load	Reads the value of a BOOL variable.	2-16
	LDN	Load NOT	Reads the inverse of the value of a BOOL variable.	2-16
	AND	AND	Takes the logical AND of the value of a BOOL variable and the input value.	2-18
	ANDN	AND NOT	Takes the logical AND of the inverse of the value of a BOOL variable and the input value.	2-18
	OR	OR	Takes the logical OR of the value of a BOOL variable and the execution condition.	2-20
	ORN	OR NOT	Takes the logical OR of the inverse of the value of a BOOL variable and the execution condition.	2-20
	Out	Output	Takes the logical result from the previous instruction and outputs it to a BOOL variable.	2-22
	OutNot	Output NOT	Takes the inverse of the logical result from the previous instruction and outputs it to a BOOL variable.	2-22
ST Statement Instructions	IF	If	Uses the evaluation result of a specified condition expression to select one of two statements to execute.	2-26
	CASE	Case	Selects the statement to execute based on the value of a specified integer expression.	2-30
	WHILE	While	Repeatedly executes a statement as long as the evaluation result of a specified condition expression is TRUE.	2-34
	REPEAT	Repeat	Executes a statement once and then executes it repeatedly until a specified condition expression is TRUE.	2-36
	EXIT	Break Loop	Used to end repeat processing from the lowest level FOR, WHILE, or REPEAT instruction.	2-38
	RETURN	Return	Ends a function or function block and returns processing to the calling instruction.	2-41
	FOR	Repeat Start	Marks the starting position for repeat processing of statements between the FOR and END_FOR statements and specifies the repeat condition.	2-42
Sequence Input Instructions	R_TRIG (Up)	Up Trigger	Outputs TRUE for one task period only when the input signal changes to TRUE.	2-44
	F_TRIG (Down)	Down Trigger	Outputs TRUE for one task period only when the input signal changes to FALSE.	2-44
	TestABit	Test A Bit	Outputs the value of the specified bit in a bit string.	2-47
	TestABitN	Test A Bit NOT	Outputs the inverse of the value of the specified bit in a bit string.	2-47
Sequence Output Instructions	RS	Reset-Priority Keep	Retains the value of a BOOL variable. It gives priority to the *Reset* input if both the *Set* input and *Reset* input are TRUE.	2-50

Type	Instruction	Name	Function	Page
Sequence Output Instructions	SR	Set-Priority Keep	Retains the value of a BOOL variable. It gives priority to the *Set* input if both the *Set* input and *Reset* input are TRUE.	2-53
	Set	Set	Changes a BOOL variable to TRUE.	2-56
	Reset	Reset	Changes a BOOL variable to FALSE.	2-56
	SetBits	Set Bits	Changes consecutive bits in bit string data to TRUE.	2-59
	ResetBits	Reset Bits	Changes consecutive bits in bit string data to FALSE.	2-59
	SetABit	Set A Bit	Changes the specified bit in bit string data to TRUE.	2-61
	ResetABit	Reset A Bit	Changes the specified bit in bit string data to FALSE.	2-61
	OutABit	Output A Bit	Changes the specified bit in bit string data to TRUE or FALSE.	2-63
Sequence Control Instructions	End	End	Ends execution of a program in the current task period.	2-66
	RETURN	Return	Ends a function or function block and returns processing to the calling instruction.	2-67
	MC	Master Control Start	Marks the starting point of a master control region and resets the master control region.	2-68
	MCR	Master Control End	Marks the end point of a master control region.	2-68
	JMP	Jump	Moves processing to the specified jump destination.	2-80
	FOR	Repeat Start	Marks the starting position for repeat processing and specifies the repeat condition.	2-82
	NEXT	Repeat End	Marks the ending position for repeat processing.	2-82
	BREAK	Break Loop	Cancels repeat processing from the lowest level FOR instruction to the NEXT instruction.	2-89
Comparison Instructions	EQ (=)	Equal	Determines if two or more values or text strings are all equivalent.	2-92
	NE (<>)	Not Equal	Determines if two values or text strings are not equivalent.	2-94
	LT (<)	Less Than	Performs a less than comparison between values.	2-97
	LE (<=)	Less Than Or Equal	Performs a less than or equal comparison between values.	2-97
	GT (>)	Greater Than	Performs a greater than comparison between values.	2-97
	GE (>=)	Greater Than Or Equal	Performs a greater than or equal comparison between values.	2-97
	EQascii	Text String Comparison Equal	Determines if two or more text strings are all equivalent.	2-100
	NEascii	Text String Comparison Not Equal	Determines if two text strings are not equivalent.	2-102
	LTascii	Text String Comparison Less Than	Performs a less than comparison between text strings.	2-104
	LEascii	Text String Comparison Less Than or Equal	Performs a less than or equal comparison between text strings.	2-104

Type	Instruction	Name	Function	Page
Comparison Instructions	GTascii	Text String Comparison Greater Than	Performs a greater than comparison between text strings.	2-104
	GEascii	Text String Comparison Greater Than or Equal	Performs a greater than or equal comparison between text strings.	2-104
	Cmp	Compare	Compares two values.	2-107
	ZoneCmp	Zone Comparison	Determines if the comparison data is within the specified maximum and minimum values.	2-109
	TableCmp	Table Comparison	Compares the comparison data with multiple defined ranges in a comparison table.	2-111
	AryCmpEQ	Array Comparison Equal	Determines if the corresponding elements of two arrays are equal.	2-114
	AryCmpNE	Array Comparison Not Equal	Determines if the corresponding elements of two arrays are not equal.	2-114
	AryCmpLT	Array Comparison Less Than	Performs a less than comparison between the corresponding elements of two arrays.	2-116
	AryCmpLE	Array Comparison Less Than Or Equal	Performs a less than or equal comparison between the corresponding elements of two arrays.	2-116
	AryCmpGT	Array Comparison Greater Than	Performs a greater than comparison between the corresponding elements of two arrays.	2-116
	AryCmpGE	Array Comparison Greater Than Or Equal	Performs a greater than or equal comparison between the corresponding elements of two arrays.	2-116
	AryCmpEQV	Array Value Comparison Equal	Determines if the elements of an array are equal to a value.	2-119
	AryCmpNEV	Array Value Comparison Not Equal	Determines if the elements of an array are not equal to a value.	2-119
	AryCmpLTV	Array Value Comparison Less Than	Performs a less than comparison between a value and the elements of an array.	2-121
	AryCmpLEV	Array Value Comparison Less Than Or Equal	Performs a less than or equal comparison between a value and the elements of an array.	2-121
	AryCmpGTV	Array Value Comparison Greater Than	Performs a greater than comparison between a value and the elements of an array.	2-121
	AryCmpGEV	Array Value Comparison Greater Than Or Equal	Performs a greater than or equal comparison between a value and the elements of an array.	2-121
Timer Instructions	TON	On-Delay Timer	Outputs TRUE when the set time elapses after the timer starts.	2-126
	TOF	Off-Delay Timer	Outputs FALSE when the set time elapses after the timer starts.	2-132
	TP	Timer Pulse	Outputs TRUE while the set time elapses after the timer starts.	2-135
	AccumulationTimer	Accumulation Timer	Totals the time that the timer input is TRUE.	2-138
	Timer	Hundred-ms Timer	Outputs TRUE when the set time elapses after the timer starts. The time is set in increments of 100 ms.	2-141

Type	Instruction	Name	Function	Page
Counter Instructions	CTD	Down-counter	Decrements the counter value when the counter input signal is received. The preset value and counter value must have an INT data type.	2-146
	CTD_**	Down-counter Group	Decrements the counter value when the counter input signal is received. The preset value and counter value must be one of the following data types: DINT, LINT, UDINT, or ULINT.	2-148
	CTU	Up-counter	Increments the counter value when the counter input signal is received. The preset value and counter value must have an INT data type.	2-150
	CTU_**	Up-counter Group	Increments the counter value when the counter input signal is received. The preset value and counter value must be one of the following data types: DINT, LINT, UDINT, or ULINT.	2-152
	CTUD	Up-down Counter	Creates an up-down counter that operates according to an up-counter input and a down-counter input. The preset value and counter value must have an INT data type.	2-155
	CTUD_**	Up-down Counter Group	Creates an up-down counter that operates according to an up-counter input and a down-counter input. The preset value and counter value must be one of the following data types: DINT, LINT, UDINT, or ULINT.	2-159
Math Instructions	ADD (+)	Addition	Adds integers and real numbers. Also joins text strings.	2-166
	AddOU (+OU)	Addition with Overflow Check	Adds integers and real numbers. Also performs an overflow check for the integer addition result.	2-170
	SUB (-)	Subtraction	Subtracts integers and real numbers.	2-174
	SubOU (-OU)	Subtraction with Overflow Check	Subtracts integers or real numbers. Also performs an overflow check for the integer subtraction result.	2-177
	MUL (*)	Multiplication	Multiplies integers and real numbers.	2-181
	MulOU (*OU)	Multiplication with Overflow Check	Multiplies integers and real numbers and outputs the result. Also performs an overflow check for the integer multiplication result.	2-185
	DIV (/)	Division	Divides integers or real numbers.	2-189
	MOD	Modulo-division	Finds the remainder for division of integers.	2-192
	ABS	Absolute Value	Finds the absolute value of an integer or real number.	2-194
	RadToDeg	Radians to Degrees	Converts a real number from radians (rad) to degrees (°).	2-196
	DegToRad	Degrees to Radians	Converts a real number from degrees (°) to radians (rad).	2-196
	SIN	Sine in Radians	Calculates the sine of a real number.	2-198
	COS	Cosine in Radians	Calculates the cosine of a real number.	2-198
	TAN	Tangent in Radians	Calculates the tangent of a real number.	2-198
	ASIN	Principal Arc Sine	Calculates the arcsine of a real number.	2-201
	ACOS	Principal Arc Cosine	Calculates the arccosine of a real number.	2-201
	ATAN	Principal Arc Tangent	Calculates the arctangent of a real number.	2-201

Type	Instruction	Name	Function	Page
Math Instructions	SQRT	Square Root	Finds the square root of a number.	2-204
	LN	Natural Logarithm	Finds the natural logarithm of a real number.	2-206
	LOG	Logarithm Base 10	Finds the base-10 logarithm of a real number.	2-206
	EXP	Natural Exponential Operation	Performs calculations for the natural exponential function.	2-209
	EXPT (**)	Exponentiation	Raises one real number to the power of another real number.	2-211
	Inc	Increment	Increments an integer value.	2-217
	Dec	Decrement	Decrements an integer value.	2-217
	Rand	Random Number	Generates pseudorandom numbers.	2-219
	AryAdd	Array Addition	Adds corresponding elements of two arrays.	2-221
	AryAddV	Array Value Addition	Adds the same value to specified elements of an array.	2-223
	ArySub	Array Subtraction	Subtracts corresponding elements of two arrays.	2-225
	ArySubV	Array Value Subtraction	Subtracts the same value from specified elements of an array.	2-227
	AryMean	Array Mean	Calculates the average of the elements of an array.	2-229
	ArySD	Array Element Standard Deviation	Calculates standard deviation of the elements of an array.	2-231
	ModReal	Real Number Modulo-division	Calculates the remainder of real number division.	2-233
	Fraction	Real Number Fraction	Finds the fractional part of a real number.	2-235
	CheckReal	Real Number Check	Checks a real number to see if it is infinity or nonnumeric data.	2-237
BCD Conversion Instructions	**_BCD_TO_***	BCD-to-Unsigned Integer Conversion Group	Converts BCD bit strings into unsigned integers.	2-242
	_TO_BCD_*	Unsigned Integer-to-BCD Conversion Group	Converts unsigned integers to BCD bit strings.	2-245
	BCD_TO_**	BCD Data Type-to-Unsigned Integer Conversion Group	Converts BCD bit strings into unsigned integers.	2-247
	BCDsToBin	Signed BCD-to-Signed Integer Conversion	Converts signed BCD bit strings to signed integers.	2-250
	BinToBCDs_**	Signed Integer-to-BCD Conversion Group	Converts signed integers to signed BCD bit strings.	2-253
	AryToBCD	Array BCD Conversion	Converts the elements of an unsigned integer array to BCD bit strings.	2-256
	AryToBin	Array Unsigned Integer Conversion	Converts the elements of an array of BCD bit strings into unsigned integers.	2-258
Data Type Conversion Instructions	**_TO_*** (Integer-to-Integer Conversion Group)	Integer-to-Integer Conversion Group	Converts integers to integers with different data types.	2-262
	_TO_* (Integer-to-Bit String Conversion Group)	Integer-to-Bit String Conversion Group	Converts integers to bit strings.	2-265

Type	Instruction	Name	Function	Page
Data Type Conversion Instructions	**_TO_*** (Integer-to-Real Number Conversion Group)	Integer-to-Real Number Conversion Group	Converts integers to real numbers.	2-268
	_TO_* (Bit String-to-Integer Conversion Group)	Bit String-to-Integer Conversion Group	Converts bit strings to integers.	2-270
	_TO_* (Bit String-to-Bit String Conversion Group)	Bit String-to-Bit String Conversion Group	Converts bit strings to bit strings with different data types.	2-272
	_TO_* (Bit String-to-Real Number Conversion Group)	Bit String-to-Real Number Conversion Group	Converts bit strings to real numbers.	2-274
	_TO_* (Real Number-to-Integer Conversion Group)	Real Number-to-Integer Conversion Group	Converts real numbers to integers.	2-276
	_TO_* (Real Number-to-Bit String Conversion Group)	Real Number-to-Bit String Conversion Group	Converts real numbers to bit strings.	2-279
	_TO_* (Real Number-to-Real Number Conversion Group)	Real Number-to-Real Number Conversion Group	Converts real numbers to real numbers with different data types.	2-281
	**_TO_STRING (Integer-to-Text String Conversion Group)	Integer-to-Text String Conversion Group	Converts integers to text strings.	2-283
	**_TO_STRING (Bit String-to-Text String Conversion Group)	Bit String-to-Text String Conversion Group	Converts bit strings to text strings.	2-285
	**_TO_STRING (Real Number-to-Text String Conversion Group)	Real Number-to-Text String Conversion Group	Converts real numbers to text strings.	2-287
	RealToFormatString	REAL-to-Formatted Text String	Converts a REAL variable to a text string with the specified format.	2-289
	LrealToFormatString	LREAL-to-Formatted Text String	Converts a LREAL variable to a text string with the specified format.	2-294
	STRING_TO_** (Text String-to-Integer Conversion Group)	Text String-to-Integer Conversion Group	Converts text strings to integers.	2-299
	STRING_TO_** (Text String-to-Bit String Conversion Group)	Text String-to-Bit String Conversion Group	Converts text strings to bit strings.	2-301
	STRING_TO_** (Text String-to-Real Number Conversion Group)	Text String-to-Real Number Conversion Group	Converts text strings to real numbers.	2-303
	TO_** (Integer Conversion Group)	Integer Conversion Group	Converts integers, bit strings, real numbers, and text strings to integers.	2-306
	TO_** (Bit String Conversion Group)	Bit String Conversion Group	Converts integers, bit strings, real numbers, and text strings to bit strings.	2-308
	TO_** (Real Number Conversion Group)	Real Number Conversion Group	Converts integers, bit strings, real numbers, and text strings to real numbers.	2-310
	EnumToNum	Enumeration-to-Integer	The EnumToNum instruction converts enumeration data to DINT data.	2-312
	NumToEnum	Integer-to-Enumeration	The NumToEnum instruction converts DINT data to enumeration data.	2-314

Type	Instruction	Name	Function	Page
Data Type Conversion Instructions	TRUNC	Truncate	Truncates a real number at the first decimal digit to make an integer.	2-316
	Round	Round Off Real Number	Rounds a real number at the first decimal digit to make an integer.	2-316
	RoundUp	Round Up Real Number	Rounds up a real number at the first decimal digit to make an integer.	2-316
Bit String Processing Instructions	AND (&)	Logical AND	Performs a logical AND operation on Boolean variables or individual bits in bit stings.	2-320
	OR	Logical OR	Performs a logical OR operation on Boolean variables or individual bits in bit stings.	2-320
	XOR	Logical Exclusive OR	Performs a logical exclusive OR operation on Boolean variables or individual bits in bit stings.	2-320
	XORN	Logical Exclusive NOR	Performs a logical exclusive NOR operation on Boolean variables or individual bits in bit stings.	2-323
	NOT	Bit Reversal	Reverses the value of a Boolean variable or individual bits in a bit string.	2-325
	AryAnd	Array Logical AND	Performs a logical AND operation on Boolean variables or individual bits in bit stings between arrays.	2-327
	AryOr	Array Logical OR	Performs a logical OR operation on Boolean variables or individual bits in bit stings between arrays.	2-327
	AryXor	Array Logical Exclusive OR	Performs a logical exclusive OR operation on Boolean variables or individual bits in bit stings between arrays.	2-327
	AryXorN	Array Logical Exclusive NOR	Performs a logical exclusive NOR operation on Boolean variables or individual bits in bit stings between arrays.	2-327
Selection Instructions	SEL	Binary Selection	Selects one of two selections.	2-332
	MUX	Multiplexer	Selects one of two to five selections.	2-334
	LIMIT	Limiter	Limits the value of the input variable to the specified minimum and maximum values.	2-337
	Band	Deadband Control	Performs deadband control.	2-339
	Zone	Dead Zone Control	Adds a bias value to the input value.	2-342
	MAX	Maximum	Finds the largest of two to five values.	2-345
	MIN	Minimum	Finds the smallest of two to five values.	2-345
	AryMax	Array Maximum	Finds the elements with the largest value in a one-dimensional array.	2-347
	AryMin	Array Minimum	Finds the elements with the smallest value in a one-dimensional array.	2-347
	ArySearch	Array Search	Searches for the specified value in a one-dimensional array.	2-350
Data Movement Instructions	MOVE	Move	Moves the value of a constant or variable to another variable.	2-354
	MoveBit	Move Bit	Moves one bit in a bit string.	2-357
	MoveDigit	Move Digit	Moves digits (4 bits per digit) in a bit string.	2-359
	TransBits	Move Bits	Moves one or more bits in a bit string.	2-361
	MemCopy	Memory Copy	Moves one or more array elements. The move source and move destination must have the same data type.	2-363

Type	Instruction	Name	Function	Page
Data Movement Instructions	SetBlock	Block Set	Moves the value of a variable or constant to one or more array elements.	2-365
	Exchange	Data Exchange	Exchanges the values of two variables.	2-367
	AryExchange	Array Data Exchange	Exchanges the elements of two arrays.	2-369
	AryMove	Array Move	Moves one or more array elements. The data types of the move source and move destina-tion can be different.	2-371
	Clear	Initialize	Initializes a variable.	2-373
	Copy**ToNum (Bit String to Signed Integer)	Bit Pattern Copy (Bit String to Signed Integer) Group	Copies the content of a bit string directly to a signed integer.	2-375
	Copy**To*** (Bit String to Real Number)	Bit Pattern Copy (Bit String to Real Number) Group	Copies the content of a bit string directly to a real number.	2-377
	CopyNumTo** (Signed Integer to Bit String)	Bit Pattern Copy (Signed Integer to Bit String) Group	Copies the content of a signed integer directly to a bit string.	2-379
	CopyNumTo** (Signed Integer to Real Num-ber)	Bit Pattern Copy (Signed Integer to Real Number) Group	Copies the content of a signed integer directly to a real number.	2-381
	Copy**To*** (Real Number to Bit String)	Bit Pattern Copy (Real Number to Bit String) Group	Copies the content of a real number directly to a bit string.	2-383
	Copy**ToNum (Real Number to Signed Integer)	Bit Pattern Copy (Real Number to Signed Integer) Group	Copies the content of a real number directly to a signed integer.	2-385
Shift Instructions	AryShiftReg	Shift Register	Shifts a bit string one bit to the left and inserts the input value to the least-significant bit. The bit string consists of array elements.	2-388
	AryShiftRegLR	Reversible Shift Register	Shifts a bit string one bit to the left or right and inserts the input value to the least-significant or most-significant bit. The bit string consists of array elements.	2-390
	ArySHL	Array N-element Left Shift	Shifts array elements by one or more elements to the left (toward the higher elements).	2-393
	ArySHR	Array N-element Right Shift	Shifts array elements by one or more elements to the right (toward the lower elements).	2-393
	SHL	N-bit Left Shift	Shifts a bit string by one or more bits to the left (toward the higher bits).	2-396
	SHR	N-bit Right Shift	Shifts a bit string by one or more bits to the right (toward the lower bits).	2-396
	NSHLC	Shift N-bits Left with Carry	Shifts an array of bit strings that includes the Carry (CY) Flag by one or more bits to the left (toward the higher elements).	2-398
	NSHRC	Shift N-bits Right with Carry	Shifts an array of bit strings that includes the Carry (CY) Flag by one or more bits to the right (toward the lower elements).	2-398
	ROL	Rotate N-bits Left	Rotates a bit string by one or more bits to the left (toward the higher bits).	2-400
	ROR	Rotate N-bits Right	Rotates a bit string by one or more bits to the right (toward the lower bits).	2-400

Type	Instruction	Name	Function	Page
Conversion Instructions	Swap	Swap Bytes	Swaps the upper byte and lower byte of a 16-bit value.	2-404
	Neg	Reverse Sign	Reverses the sign of a number.	2-405
	Decoder	Bit Decoder	Sets the specified bit to TRUE and the other bits to FALSE in array elements that consist of a maximum of 256 bits.	2-407
	Encoder	Bit Encoder	Finds the position of the highest TRUE bit in array elements that consist of a maximum of 256 bits.	2-410
	BitCnt	Bit Counter	Counts the number of TRUE bits in a bit string.	2-412
	ColmToLine_**	Column to Line Conversion Group	Extracts bit values from the specified position of array elements and outputs them as a bit string.	2-413
	LineToColm	Line to Column Conversion	Takes the bits from a bit string and outputs them to the specified bit position in array elements.	2-415
	Gray	Gray Code Conversion	Converts a gray code into an angle.	2-417
	UTF8ToSJIS	UTF-8 to SJIS Character Code Conversion	Converts a UTF-8 text string to a SJIS BYTE array.	2-422
	SJISToUTF8	SJIS to UTF-8 Character Code Conversion	Converts a SJIS BYTE array to a UTF-8 text string.	2-424
	PWLApprox	Broken Line Approximation with Broken Line Data Check	Performs broken line approximations for integers or real numbers along with a check of the validity of the broken line data.	2-426
	PWLApproxNoLineChk	Broken Line Approximation without Broken Line Data Check	Performs broken line approximations for integers or real numbers along without a check of the validity of the broken line data.	2-426
	PWLLineChk	Broken Line Data Check	Checks whether the X coordinates in the broken line data that is used for a Broken Line Approximation without Broken Line Data Check instruction are in ascending order.	2-432
	MovingAverage	Moving Average	Calculates a moving average.	2-435
	DispartReal	Separate Mantissa and Exponent	Separates a real number into the signed mantissa and the exponent.	2-441
	UniteReal	Combine Real Number Mantissa and Exponent	Combines a signed mantissa and exponent to make a real number.	2-444
	NumToDecString	Fixed-length Decimal Text String Conversion	Converts an integer to a fixed-length decimal text string.	2-446
	NumToHexString	Fixed-length Hexadecimal Text String Conversion	Converts an integer to a fixed-length hexadecimal text string.	2-446
	HexStringToNum_**	Hexadecimal Text String-to-Number Conversion Group	Converts a hexadecimal text string to an integer.	2-449
	FixNumToString	Fixed-decimal Number-to-Text String Conversion	Converts a signed fixed-decimal number to a decimal text string.	2-451

Type	Instruction	Name	Function	Page
Conversion Instructions	StringToFixNum	Text String-to-Fixed-decimal Conversion	Converts a decimal text string to a signed fixed-decimal number.	2-453
	DtToString	Date and Time-to-Text String Conversion	Converts a date and time to a text string.	2-456
	DateToString	Date-to-Text String Conversion	Converts a date to a text string.	2-458
	TodToString	Time of Day-to-Text String Conversion	Converts a time of day to a text string.	2-459
	GrayToBin_**	Gray Code-to-Binary Code Conversion Group	Converts a gray code to a bit string.	2-461
	BinToGray_**	Binary Code-to-Gray Code Conversion	Converts a bit string to a gray code.	2-461
	StringToAry	Text String-to-Array Conversion	Converts a text string to a BYTE array.	2-463
	AryToString	Array-to-Text String Conversion	Converts a BYTE array to a text string.	2-465
	DispartDigit	Four-bit Separation	Separates a bit string into 4-bit units.	2-467
	UniteDigit_**	Four-bit Join Group	Joins 4-bit units of data into a bit string.	2-469
	Dispart8Bit	Byte Data Separation	Separates a bit string into individual bytes.	2-471
	Unite8Bit_**	Byte Data Join Group	Joins bytes of data into a bit string.	2-473
	ToAryByte	Conversion to Byte Array	Separates the value of a variable into bytes and stores them in a BYTE array.	2-475
	AryByteTo	Conversion from Byte Array	Joins BYTE array elements and stores the result in a variable.	2-480
	SizeOfAry	Get Number of Array Elements	Gets the number of elements in an array.	2-485
	PackWord	2-byte Join	Joins two 1-byte data into a 2-byte data.	2-487
	PackDword	4-byte Join	Joins four 1-byte data into a 4-byte data.	2-489
	LOWER_BOUND	Get First Number of Array	Gets the first number of array dimensions.	2-491
	UPPER_BOUND	Get Last Number of Array	Gets the last number of array dimensions.	2-491

Type	Instruction	Name	Function	Page
Stack and Table Instructions	StackPush	Push onto Stack	Stores a value in a stack.	2-498
	StackFIFO	First In First Out	Removes the bottom value from a stack.	2-507
	StackLIFO	Last In First Out	Removes the top value from a stack.	2-507
	StackIns	Insert into Stack	Inserts a value at a specified position in a stack.	2-510
	StackDel	Delete from Stack	Deletes a value from a specified position in a stack.	2-512
	RecSearch	Record Search	Searches an array of structures for elements that match the search key with the specified method.	2-514
	RecRangeSearch	Range Record Search	Searches an array of structures for elements that match the search condition range with the specified method.	2-519
	RecSort	Record Sort	Sorts the elements of an array of structures.	2-524
	RecNum	Get Number of Records	Finds the number of records in an array of structures to the end data.	2-530
	RecMax	Maximum Record Search	Searches the specified member in the structures of an array of structures for the maximum value.	2-532
	RecMin	Minimum Record Search	Searches the specified member in the structures of an array of structures for the minimum value.	2-532
FCS Instructions	StringSum	Checksum Calculation	Calculates the checksum for a text string.	2-538
	StringLRC	Calculate Text String LRC	Calculates the LRC value (horizontal parity).	2-540
	StringCRCCCITT	Calculate Text String CRC-CCITT	Calculates the CRC-CCITT value using the XMODEM method.	2-542
	StringCRC16	Calculate Text String CRC-16	Calculates the CRC-16 value using the MODBUS method.	2-544
	AryLRC_**	Calculate Array LRC Group	Calculates the LRC value for an array	2-546
	AryCRCCCITT	Calculate Array CRC-CCITT	Calculates the CRC-CCITT value using the XMODEM method.	2-548
	AryCRC16	Calculate Array CRC-16	Calculates the CRC-16 value using the MODBUS method.	2-550

Type	Instruction	Name	Function	Page
Text String Instructions	CONCAT	Concatenate String	Joins two to five text strings.	2-554
	LEFT	Get String Left	Extracts a text string with the specified number of characters from the start (left) of a text string.	2-556
	RIGHT	Get String Right	Extracts a text string with the specified number of characters from the end (right) of a text string.	2-556
	MID	Get String Any	Extracts a text string with the specified number of characters from the specified character position.	2-558
	FIND	Find String	Searches a specified text string for the position of a specified text string.	2-560
	LEN	String Length	Finds the number of characters in a text string.	2-562
	REPLACE	Replace String	Replaces part of a text string with another text string	2-563
	DELETE	Delete String	Deletes all or part of a text string.	2-565
	INSERT	Insert String	Inserts a text string into another text string.	2-567
	GetByteLen	Get Byte Length	Counts the number of bytes in a text string.	2-569
	ClearString	Clear String	Clears a text string.	2-571
	ToUCase	Convert to Uppercase	Converts all single-byte letters in a text string to uppercase.	2-573
	ToLCase	Convert to Lowercase	Converts all single-byte letters in a text string to lowercase.	2-573
	TrimL	Trim String Left	Removes blank space from the beginning of a text string.	2-575
	TrimR	Trim String Right	Removes blank space from the end of a text string.	2-575
	AddDelimiter	Put Text Strings with Delimiters	Converts the values in a structure to text strings and adds delimiters.	2-577
	SubDelimiter	Get Text Strings Minus Delimiters	Reads delimited data from a text string and stores the results as the values of the mem-bers of a structure.	2-588

Type	Instruction	Name	Function	Page
Time and Time of Day Instructions	ADD_TIME	Add Time	Adds two times.	2-600
	ADD_TOD_TIME	Add Time to Time of Day	Adds a time to a time of day.	2-602
	ADD_DT_TIME	Add Time to Date and Time	Adds a time to a date and time.	2-604
	SUB_TIME	Subtract Time	Subtracts one time from another.	2-606
	SUB_TOD_TIME	Subtract Time from Time of Day	Subtracts a time from a time of day.	2-608
	SUB_TOD_TOD	Subtract Time of Day	Subtracts a time of day from another time of day.	2-610
	SUB_DATE_DATE	Subtract Date	Subtracts another date from another date.	2-611
	SUB_DT_DT	Subtract Date and Time	Subtracts another date and time from another date and time.	2-612
	SUB_DT_TIME	Subtract Time from Date and Time	Subtracts a time from a date and time.	2-614
	MULTIME	Multiply Time	Multiplies a time by a specified number.	2-616
	DIVTIME	Divide Time	Divides a time by a specified number.	2-618
	CONCAT_DATE_TOD	Concatenate Date and Time of Day	Combines a date and a time of day.	2-620
	DT_TO_TOD	Extract Time of Day from Date and Time	Extracts the time of day from a date and time.	2-622
	DT_TO_DATE	Extract Date from Date and Time	Extracts the date from a date and time.	2-624
	GetTime	Get Time of Day	Reads the current time.	2-626
	DtToSec	Convert Date and Time to Seconds	Converts a date and time to the number of seconds from 00:00:00 on January 1, 1970.	2-628
	DateToSec	Convert Date to Seconds	Converts a date to the number of seconds from 00:00:00 on January 1, 1970.	2-630
	TodToSec	Convert Time of Day to Seconds	Converts a time of day to the number of seconds from 00:00:00.	2-631
	SecToDt	Convert Seconds to Date and Time	Converts the number of seconds from 00:00:00 on January 1, 1970 to a date and time.	2-632
	SecToDate	Convert Seconds to Date	Converts the number of seconds from 00:00:00 on January 1, 1970 to a date.	2-634
	SecToTod	Convert Seconds to Time of Day	Converts the number of seconds from 00:00:00 to a time of day.	2-636
	TimeToNanoSec	Convert Time to Nanoseconds	Converts a time to nanoseconds.	2-638
	TimeToSec	Convert Time to Seconds	Converts a time to seconds.	2-639
	NanoSecToTime	Convert Nanoseconds to Time	Converts nanoseconds to a time.	2-640
	SecToTime	Convert Seconds to Time	Converts seconds to a time.	2-641
	ChkLeapYear	Check for Leap Year	Checks for a leap year.	2-643
	GetDaysOfMonth	Get Days in Month	Gets the number of days in the specified month.	2-644

Type	Instruction	Name	Function	Page
Time and Time of Day Instructions	DaysToMonth	Convert Days to Month	Calculates the month based on the number of days from January 1.	2-646
	GetDayOfWeek	Get Day of Week	Gets the day of the week for the specified year, month, and day of month.	2-648
	GetWeekOfYear	Get Week Number	Gets the week number for the specified year, month, and day of month.	2-650
	DtToDateStruct	Break Down Date and Time	Converts a date and time to the year, month, day, hour, minutes, seconds, and nanoseconds.	2-652
	DateStructToDt	Join Time	Joins a year, month, day, hour, minutes, seconds, and nanoseconds into a date and time.	2-655
	TruncTime	Truncate Time	Truncates a TIME variable below the specified time unit.	2-657
	TruncDt	Truncate Date and Time	Truncates a DT variable below the specified time unit.	2-661
	TruncTod	Truncate Time of Day	Truncates a TOD variable below the specified time unit.	2-665
Analog Control Instructions	PIDAT	PID Control with Autotuning	Performs PID control with autotuning (2-PID control with set point filter).	2-670
	PIDAT_HeatCool	Heating/Cooling PID with Autotuning	Performs heating/cooling PID control with autotuning (2-PID control with set point filter).	2-695
	TimeProportionalOut	Time-proportional Output	Converts a manipulated variable to a time-proportional output.	2-733
	LimitAlarm_**	Upper/Lower Limit Alarm Group	Outputs an alarm if the input value is below the lower limit set value or above the upper limit set value.	2-750
	LimitAlarmDv_**	Upper/Lower Deviation Alarm Group	Outputs an alarm if the deviation in the input value from the reference value exceeds the lower deviation set value or the upper deviation set value.	2-754
	LimitAlarmDvStbySeq_**	Upper/Lower Deviation Alarm with Standby Sequence Group	Outputs upper and lower deviation alarms with a standby sequence.	2-759
	ScaleTrans	Scale Transformation	Converts input values from an input range to an output range.	2-774
	AC_StepProgram	Step Program	Calculates the present set point and the predicted set point every task period according to the specified program pattern.	2-777
System Control Instructions	TraceSamp	Data Trace Sampling	Performs sampling for a data trace.	2-804
	TraceTrig	Data Trace Trigger	Generates a trigger for data tracing.	2-807
	GetTraceStatus	Read Data Trace Status	Reads the execution status of a data trace.	2-810
	SetAlarm	Create User-defined Error	Creates a user-defined error.	2-814
	ResetAlarm	Reset User-defined Error	Resets a user-defined error.	2-819
	GetAlarm	Get User-defined Error Status	Gets the highest event level (of user-defined error levels 1 to 8) and the highest level event code of the current user-defined errors.	2-821
	ResetPLCError	Reset PLC Controller Error	Resets errors in the PLC Function Module.	2-823

Type	Instruction	Name	Function	Page
System Control Instructions	GetPLCError	Get PLC Controller Error Status	Gets the highest level status (partial fault or minor fault) and highest level event code of the current Controller errors in the PLC Function Module.	2-826
	GetEIPError	Get EtherNet/IP Error Status	Gets the highest level status (partial fault or minor fault) and highest level event code of the current Controller errors in the EtherNet/IP Function Module.	2-828
	ResetMCError	Reset Motion Control Error	Resets a Controller Error in the Motion Control Function Module.	2-830
	GetMCError	Get Motion Control Error Status	Gets the highest level status (partial fault or minor fault) and highest level event code of the current Controller errors in the Motion Control Function Module.	2-835
	ResetECError	Reset EtherCAT Error	Resets a Controller Error in the EtherCAT Master Function Module.	2-837
	GetECError	Get EtherCAT Error Status	Detects errors in the EtherCAT Master Function Module.	2-839
	SetInfo	Create User-defined Information	Creates user-defined information.	2-842
	RestartNXUnit	Restart NX Unit	Restarts an EtherCAT Coupler Unit or NX Unit.	2-844
	NX_ChangeWriteMode	Change to NX Unit Write Mode	Changes an EtherCAT Coupler Unit or NX Unit to a mode that allows writing data.	2-851
	NX_SaveParam	Save NX Unit Parameters	Saves the data that was written to an EtherCAT Coupler Unit or NX Unit.	2-856
	NX_ReadTotalPowerOnTime	Read NX Unit Total Power ON Time	Reads the total power ON time from a Communications Coupler Unit or NX Unit.	2-862
Program Control Instructions	PrgStart	Enable Program	Enables the execution of the specified program.	2-872
	PrgStop	Disable Program	Disables execution of the specified program.	2-881
	PrgStatus	Read Program Status	Reads the status of the specified program.	2-901

Type	Instruction	Name	Function	Page
EtherCAT Communications Instructions	EC_CoESDOWrite	Write EtherCAT CoE SDO	Writes a value to a CoE object of a specified slave on the EtherCAT network.	2-908
	EC_CoESDORead	Read EtherCAT CoE SDO	Reads a value from a CoE object of a specified slave on the EtherCAT network.	2-911
	EC_StartMon	Start EtherCAT Packet Monitor	Starts packet monitoring for EtherCAT communications.	2-916
	EC_StopMon	Stop EtherCAT Packet Monitor	Stops execution of packet monitoring for EtherCAT communications.	2-922
	EC_SaveMon	Save EtherCAT Packets	Saves EtherCAT communications packet data to an internal file in the main memory of the CPU Unit.	2-924
	EC_CopyMon	Transfer EtherCAT Packets	Transfers packet data in an internal file in the main memory of the CPU Unit to a SD Memory Card.	2-926
	EC_DisconnectSlave	Disconnect EtherCAT Slave	Disconnects the specified slave from the EtherCAT network.	2-928
	EC_ConnectSlave	Connect EtherCAT Slave	Connects the specified slave to the EtherCAT network.	2-935
	EC_ChangeEnableSetting	Enable/Disable EtherCAT Slave	Enables or disables an EtherCAT slave.	2-937
	NX_WriteObj	Write NX Unit Object	Writes data to an NX object in an EtherCAT Coupler Unit or NX Unit.	2-954
	NX_ReadObj	Read NX Unit Object	Reads data from an NX object in an EtherCAT Coupler Unit or NX Unit.	2-969
IO-Link Communications Instruction	IOL_ReadObj	Read IO-Link Device Object	Reads data from IO-Link device objects.	2-978
	IOL_WriteObj	Write IO-Link Device Object	Writes data to IO-Link device objects.	2-987
EtherNet/IP Communications Instructions	CIPOpen	Open CIP Class 3 Connection (Large_Forward_Open)	Opens a CIP class 3 connection (Large_Forward_Open) with the specified remote node. The data length is set to 1,994 bytes.	2-998
	CIPOpenWithDataSize	Open CIP Class 3 Connection with Specified Data Size	Opens a CIP class 3 connection with the specified remote node that allows class 3 explicit messages of the specified data length or shorter to be sent and received.	2-1007
	CIPRead	Read Variable Class 3 Explicit	Uses a class 3 explicit message to read the value of a variable in another Controller on a CIP network.	2-1011
	CIPWrite	Write Variable Class 3 Explicit	Uses a class 3 explicit message to write the value of a variable in another Controller on a CIP network.	2-1017
	CIPSend	Send Explicit Message Class 3	Sends a class 3 CIP message to a specified device on a CIP network.	2-1023
	CIPClose	Close CIP Class 3 Connection	Closes the CIP class 3 connection to the specified handle.	2-1028
	CIPUCMMRead	Read Variable UCMM Explicit	Uses a UCMM explicit message to read the value of a variable in another Controller on the specified CIP network.	2-1031
	CIPUCMMWrite	Write Variable UCMM Explicit	Uses a UCMM explicit message to write the value of a variable in another Controller on a CIP network.	2-1036
	CIPUCMMSend	Send Explicit Message UCMM	Sends a UCMM CIP message to a specified device on a CIP network.	2-1043

Type	Instruction	Name	Function	Page
EtherNet/IP Communica-tions Instruc-tions	SktUDPCreate	Create UDP Socket	Creates a UDP socket request to open a servo port for the built-in EtherNet/IP.	2-1053
	SktUDPRcv	UDP Socket Receive	Reads the data from the receive buffer for a UDP socket for the built-in EtherNet/IP.	2-1061
	SktUDPSend	UDP Socket Send	Sends data from a UDP port for the built-in EtherNet/IP.	2-1064
	SktTCPAccept	Accept TCP Socket	Requests accepting a TCP socket for the built-in EtherNet/IP.	2-1067
	SktTCPConnect	Connect TCP Socket	Connects to a remote TCP port from the built-in EtherNet/IP.	2-1070
	SktTCPRcv	TCP Socket Receive	Reads the data from the receive buffer for a TCP socket for the built-in EtherNet/IP.	2-1079
	SktTCPSend	TCP Socket Send	Sends data from a TCP port for the built-in EtherNet/IP.	2-1082
	SktGetTCPStatus	Read TCP Socket Status	Reads the status of a TCP socket.	2-1085
	SktClose	Close TCP/UDP Socket	Closes the specified TCP or UDP socket for the built-in EtherNet/IP.	2-1088
	SktClearBuf	Clear TCP/UDP Socket Receive Buffer	Clears the receive buffer for the specified TCP or UDP socket for the built-in EtherNet/IP.	2-1091
	SktSetOption	Set TCP Socket Option	Sets the option for TCP socket specified for the built-in EtherNet/IP.	2-1094
	ChangeIPAdr	Change IP Address	Changes the IP address of the built-in EtherNet/IP port or the IP address of an EtherNet/IP Unit.	2-1099
	ChangeFTPAccount	Change FTP Account	Changes the FTP login name and password of the built-in EtherNet/IP port or those of an EtherNet/IP Unit.	2-1107
	FTPGetFileList	Get FTP Server File List	Gets a list of the files in the FTP server.	2-1111
	FTPGetFile	Get File from FTP Server	Downloads a file from the FTP server.	2-1128
	FTPPutFile	Put File onto FTP Server	Uploads a file to the FTP server.	2-1137
	FTPRemoveFile	Delete FTP Server File	Deletes a file from the FTP server.	2-1148
	FTPRemoveDir	Delete FTP Server Directory	Deletes a directory from the FTP server.	2-1158

Type	Instruction	Name	Function	Page
Serial Communications Instructions	NX_SerialSend	Send No-protocol Data	Sends data in No-protocol Mode from a serial port on an NX-series Communications Interface Unit or Option Board.	2-1164
	NX_SerialRcv	Receive No-protocol Data	Reads data in No-protocol Mode from a serial port on an NX-series Communications Interface Unit or Option Board.	2-1177
	NX_ModbusRtuCmd	Send Modbus RTU General Command	Sends general commands from a serial port on an NX-series Communications Interface Unit or Option Board to Modbus-RTU slaves using Modbus-RTU protocol.	2-1191
	NX_ModbusRtuRead	Send Modbus RTU Read Command	Sends read commands from a serial port on an NX-series Communications Interface Unit or Option Board to Modbus-RTU slaves using Modbus-RTU protocol.	2-1202
	NX_ModbusRtuWrite	Send Modbus RTU Write Command	Sends write commands from a serial port on an NX-series Communications Interface Unit or Option Board to Modbus-RTU slaves using Modbus-RTU protocol.	2-1214
	NX_SerialSigCtl	Serial Control Signal ON/OFF Switching	Turns ON or OFF the ER or RS signal of a serial port on an NX-series Communications Interface Unit or Option Board.	2-1226
	NX_SerialBufClear	Clear Buffer	Clears the send or receive buffer.	2-1235
	NX_SerialStartMon	Start Serial Line Monitoring	Starts serial line monitoring of an NX-series Communications Interface Unit.	2-1245
	NX_SerialStopMon	Stop Serial Line Monitoring	Stops serial line monitoring of an NX-series Communications Interface Unit.	2-1250

Type	Instruction	Name	Function	Page
SD Memory Card Instructions	FileWriteVar	Write Variable to File	Writes the value of a variable to the specified file in the SD Memory Card. The value is written in binary format.	2-1256
	FileReadVar	Read Variable from File	Reads the contents of the specified file on the SD Memory Card as binary data and writes it to a variable.	2-1261
	FileOpen	Open File	Opens the specified file in the SD Memory Card.	2-1266
	FileClose	Close File	Closes the specified file in the SD Memory Card.	2-1270
	FileSeek	Seek File	Sets a file position indicator in the specified file in the SD Memory Card.	2-1273
	FileRead	Read File	Reads the data from the specified file in the SD Memory Card.	2-1277
	FileWrite	Write File	Writes data to the specified file in the SD Memory Card.	2-1285
	FileGets	Get Text String	Reads a text string of one line from the specified file in the SD Memory Card.	2-1293
	FilePuts	Put Text String	Writes a text string to the specified file in the SD Memory Card.	2-1301
	FileCopy	Copy File	Copies the specified file in the SD Memory Card.	2-1310
	FileRemove	Delete File	Deletes the specified file from the SD Memory Card.	2-1319
	FileRename	Change File Name	Changes the name of the specified file or directory in the SD Memory Card.	2-1324
	DirCreate	Create Directory	Creates a directory with the specified name in the SD Memory Card.	2-1329
	DirRemove	Delete Directory	Deletes the specified directory from the SD Memory Card.	2-1332
	BackupToMemoryCard	SD Memory Card Backup	Backs up data to an SD Memory Card.	2-1335
Time Stamp Instructions	NX_DOutTimeStamp	Write Digital Output with Specified Time Stamp	Writes a value to the output bit of a Digital Output Unit that supports time stamp refreshing.	2-1352
	NX_AryDOutTimeStamp	Write Digital Output Array with Specified Time Stamp	Outputs pulses from a Digital Output Unit that supports time stamp refreshing.	2-1358
OS Control Instructions	IPC_GetOSStatus	Read OS Status	Reads the status of the Industrial PC's operating system (Windows).	2-1368
	IPC_RebootOS	Restart OS	Restarts the Industrial PC's operating system (Windows).	2-1371
	IPC_Shutdown	Shut Down	Starts the shutdown of the Industrial PC and, when completed, notifies Windows of the shutdown.	2-1374
Other Instructions	ReadNbit_**	N-bit Read Group	Reads zero or more bits from a bit string.	2-1378
	WriteNbit_**	N-bit Write Group	Writes zero or more bits to a bit string.	2-1380
	ChkRange	Check Subrange Variable	Determines if the value of a variable is within the valid range of the range type specification.	2-1382
	GetMyTaskStatus	Read Current Task Status	Reads the status of the current task.	2-1384

Type	Instruction	Name	Function	Page
	GetMyTaskInterval	Read Current Task Period	Reads the task period of the current task.	2-1387
	Task_IsActive	Determine Task Status	Determines if the specified task is currently in execution.	2-1390
	Lock	Lock Tasks	Starts an exclusive lock between tasks. Execution of any other task with a lock region with the same lock number is disabled.	2-1392
	Unlock	Unlock Tasks	Stops an exclusive lock between tasks.	2-1392
	ActEventTask	Activate Event Task	Activates an event task.	2-1399
	Get**Clk	Get Clock Pulse Group	Outputs a clock pulse at the specified cycle.	2-1405
	Get**Cnt	Get Incrementing Free-running Counter Group	Gets the values of free-running counters of the specified cycle.	2-1406

- Refer to the *NY-series Motion Control Instructions Reference Manual* (Cat. No. W561) for the specifications of the motion control instructions.
- Refer to the *Sysmac Studio Version 1 Operation Manual* (Cat. No. W504) for the specifications of the simulation instructions.

2

Instruction Descriptions

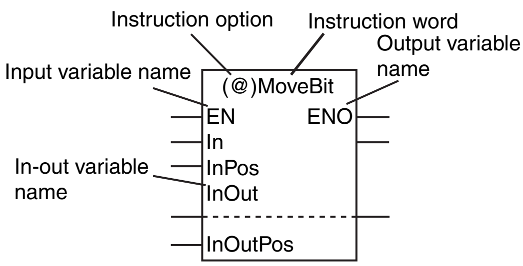
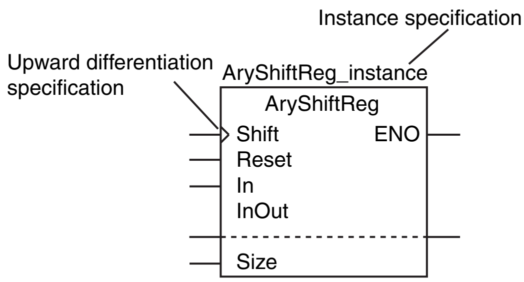
This section describes the specifications of the instructions that you can use with NY-series Controllers.

Using this Section

The notation used to describe instructions in this section is explained below.

Items

The following items are provided.

Item	Description
Instruction	The instruction word is given. Example: MoveBit
Name	The name of the instruction is given. Example: Move Bit
FB/FUN	Whether the instruction is a function block (FB) instruction or a function (FUN) instruction is given. You can call FB instructions only from programs and function blocks. You can call FUN instructions from programs, function blocks, and functions.
Graphic expression	The figure that represents the instruction in a ladder diagram is given. **Example for a FUN Instruction**  **Example for a FB Instruction**  The instruction option, upward differentiation specification, and instance specification are described below. Instruction option: Support for the instruction option is indicated by "(@)" before the FUN instruction. If support for the instruction option is indicated, you can place "@" before the instruction word to specify upward differentiation. An instruction for which upward differentiation is specified is executed when the value of the EN input variable was FALSE in the previous task period and is TRUE in the current task period. Upward differentiation specification: This is indicated by the arrow pointing into the instruction at the entry point of the input variable. Instructions with this specification operate as upwardly differentiated instructions. Instance specification: An instance of an instruction is indicated by "XX_instance" above an FB instruction. You must assign an instance name to any instance of an instruction that you specify.
ST expression	The notation that represents the instruction in ST is given. There are two ways that you can use to code an instruction in ST. These are described below. 1. Directly Specifying the Correspondence between the Parameters and the Input, Output, and In-Out Variables Example: MoveBit(In:=abc, InPos:=def, InOut:=ghi, InOutPos:=jkl); 2. Specifying Only the Parameters and Omitting the Input, Output, and In-Out Variables Example: MoveBit(In, InPos, InOut, InOutPos); Method 2 is used in this section. You must assign an instance name to any instruction that is given as "XX_instance(*variable_name*)." Example: TON_instance (In, PT, Q, ET);

Item	Description
Variables	• Name The input variables, output variables, and in-out variables are given. Example: *In1* However, variables that are used by many instructions are not given on the pages that describe individual instructions. The following eight variables are commonly used. The specifications of these variables are given later. (*EN*, *ENO*, *Execute*, *Done*, *Busy*, *Error*, *ErrorID*, and *ErrorIDEx*) • Meaning The name of the variable is given. Example: Up-counter • I/O Whether the variable is an input variable, output variable, or in-out variable is given. • Description The meaning of the variable and any restrictions are given. • Valid range The range that the variable can take is given. “Depends on data type” indicates that the valid range of the variable depends on the data type that you use. The valid ranges of the data types are given later in this section. • Unit The unit of the value that is specified with the variable is given. “---” indicates that there is no unit. Example: Bytes • Default The specified default value is automatically used for the variable if you do not assign a parameter to the instruction before it is executed. “---” indicates the following: Input variables: The default value of the data type of the input variable is assigned. The default values of the data types are given later in this section. If the input variable is a structure, the default value is given in the specifications of the structure in the description of the function of the instruction. Output variables: Default values are not set. In-out variables: Default values are not set. • Data type The data type of the variable is given. The use of enumerations, arrays, structures, and unions is also given.
Function	The function of the instruction is described. Variable names are given in italic text. Example: *In1* Array names are followed by “[]”. Example: *InOut[]*
Related System-defined Variables	The system-defined variables that are related to the instruction are given. Refer to the *NJ/NX-series CPU Unit Software User's Manual* (Cat. No. W501) or *NY-series Industrial Panel PC / Industrial Box PC Software User’s Manual* (Cat. No. W558) for details on system-defined variables.
Related Semi-user-defined Variables	The semi-user-defined variables and variable names that are related to the instruction are given. Refer to the specified manuals for details on semi-user-defined variables.
Additional Information	Additional information on the function of the instruction is provided. This includes related instructions and helpful information for application of the instruction.
Precautions for Correct Use	Precautions for application of the instruction are given. The conditions under which errors occur for the instruction are also given here.
Sample Programming	Short samples of how to use the instruction in an application program are provided. The ladder diagram and ST for the same process are shown.

Common Variables

The specifications of variables that are used for many instructions (*EN, ENO, Execute, Done, Busy, Error, ErrorID,* and *ErrorIDEx*) are described below. These variables are not described in the tables of variables for individual instructions. Check the graphic or ST expression for the instruction to see if an instruction uses these variables.

EN

EN is an input variable that gives the execution condition for a FUN instruction.
When you use a FUN instruction in a ladder diagram, connect the execution condition to *EN*.

Name	Meaning	I/O	Description	Data type	Valid range	Default
EN	Enable (Execution Condition)	Input	TRUE: Instruction is executed.* FALSE: Instruction is not executed.	BOOL	TRUE or FALSE	TRUE

* If upward differentiation (@) is specified as an instruction option, the execution condition is when the value of *EN* changes from FALSE to TRUE. If downward differentiation (%) is specified as an instruction option, the execution condition is when the value of *EN* changes from TRUE to FALSE.

- FB instructions do not have an *EN* input variable.
- When you call a FUN instruction from structured text, omit the *EN* input variable. The *EN* input variable is not required in structured text because the execution condition for the instruction is determined by the operation sequence.

ENO

The *ENO* output variable passes the execution to the next instruction in a ladder diagram. Normally, when instruction execution is completed, the value of *ENO* changes to TRUE. Execution of the next instruction is then started.

Name	Meaning	I/O	Description	Data type	Valid range	Default
ENO	Enable Output	Output	TRUE: Normal end.* FALSE: Error end, execution in progress, or execution condition not met.	BOOL	TRUE or FALSE	---

* *ENO* is TRUE only while the execution condition is met. The value of *ENO* changes to FALSE when the execution condition is no longer met after a normal end.

- Most FUN instructions and FB instructions have *ENO* output variables. There are, however, some instructions that do not have an *ENO* output variable.
- Omit the *ENO* output variable in structured text. The *ENO* output variable is not required in structured text because the execution condition for the next instruction is determined by the operation sequence.

Execute, *Done*, and *Busy*

Execute is an input variable that gives the execution condition for some FB instructions.
Instruction execution starts when *Execute* changes to TRUE. After *Execute* changes to TRUE, execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the instruction execution time exceeds the task period.

Done is an output variable that shows the completion of execution for some FB instructions.
Busy is an output variable that shows that instruction execution is in progress for some FB instructions.

Name	Meaning	I/O	Description	Data type	Valid range	Initial value
Execute	Execute	Input	TRUE: Instruction is executed.*1 FALSE: Instruction is not executed.*2	BOOL	TRUE or FALSE	FALSE
Done	Done	Output	TRUE: Normal end.*3*4 FALSE: Error end, execution in progress, or execution condition not met.	BOOL	TRUE or FALSE	---
Busy	Busy		TRUE: Execution processing is in progress. FALSE: Execution processing is not in progress.			

*1 If the value of *Execute* is already TRUE when Controller operation starts, the instruction is not executed. To execute the instruction in that case, first change the value of *Execute* to FALSE.

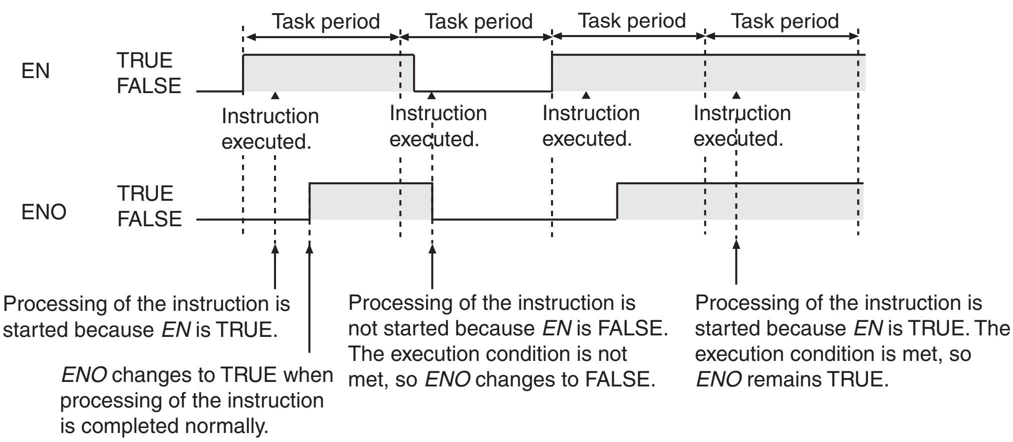
*2 Processing is completed to the end even if *Execute* changes to FALSE during execution.

*3 The value of *Done* changes to FALSE when the execution condition is no longer met after a normal end.

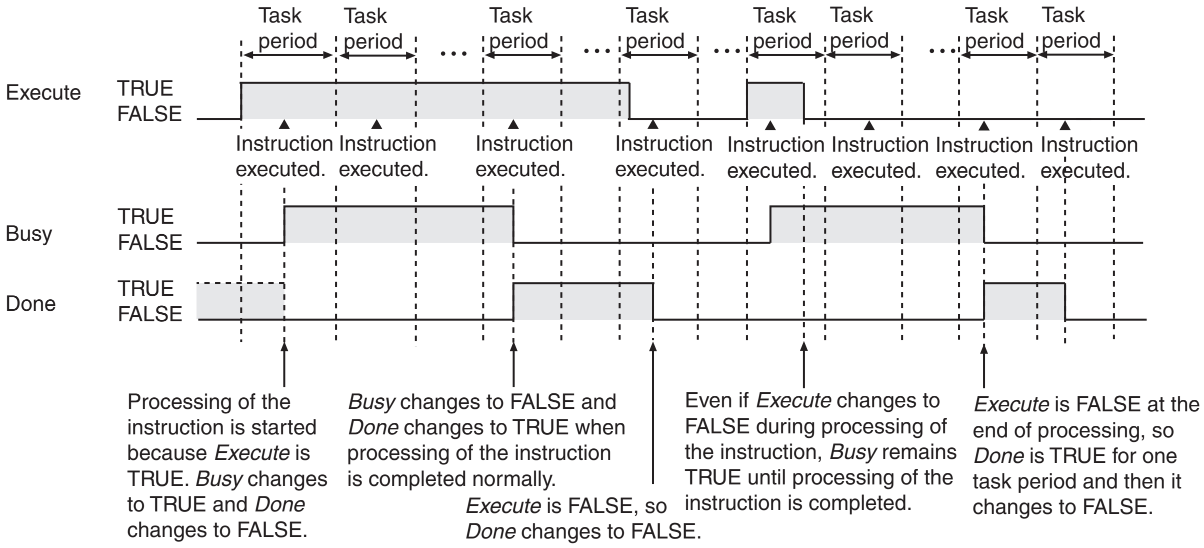
*4 If the execution condition is no longer met when a normal end occurs, the value of *Done* is TRUE for one task period and it then changes to FALSE.

Timing charts are given below for instructions that have *EN* and *ENO* variables (i.e., instructions that are completed in one task period) and for instructions that have *Execute* and *Busy* variables (i.e., instructions that are processed over more than one task period).

● Instructions Completed in One Task Period



● Instructions Processed over More Than One Task Period



Error, ErrorID, and *ErrorIDEx*

Error, *ErrorID*, and *ErrorIDEx* are output variables that show that an error occurred in the execution of some FB instructions.

Name	Meaning	I/O	Description	Data type	Valid range	Initial value
Error	Error	Output	TRUE: Error end.*1*2 FALSE: Normal end, execution in progress, or execution condition not met.	BOOL	TRUE or FALSE	---
ErrorID	Error code		This is the error ID for an error end. The value is WORD#16#0 for a normal end.	WORD	Depends on the instruction.	
ErrorIDEx	Expansion error code		This is the error ID for an Expansion Unit Hardware Error. The value is DWORD#16#0 for a normal end.	DWORD		

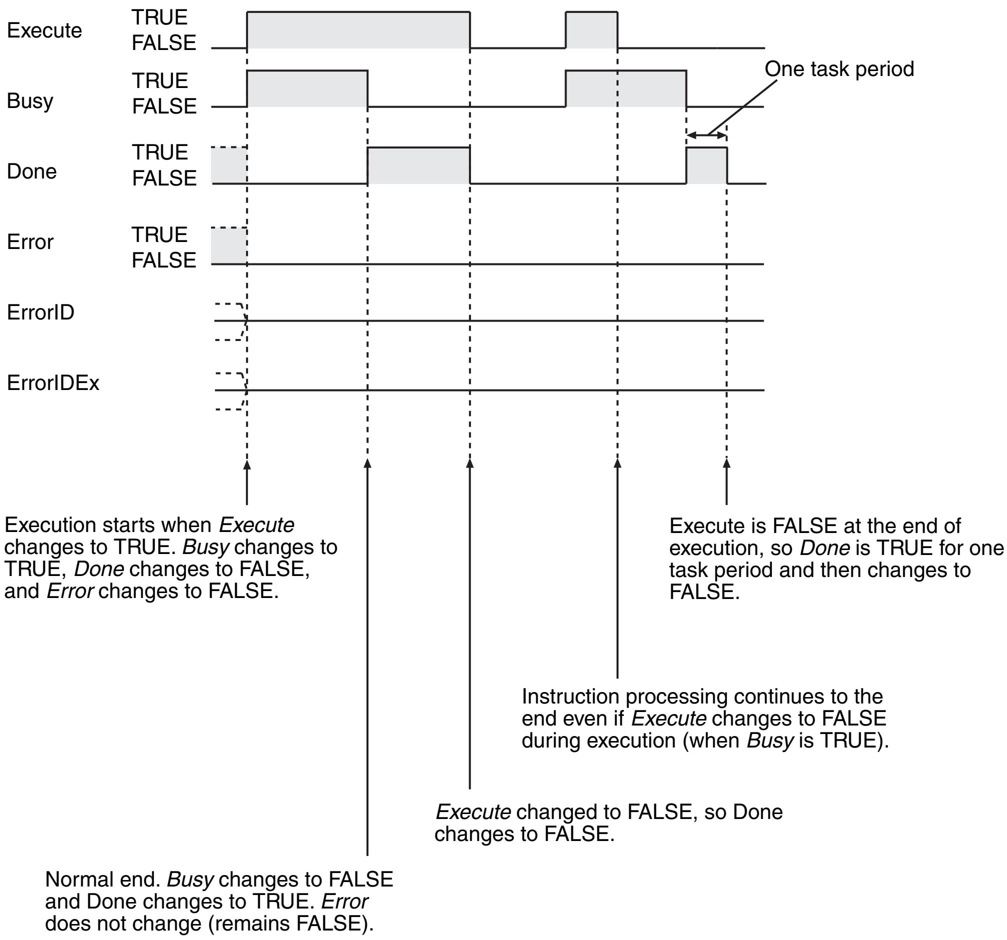
*1 The value of *Error* changes to FALSE when the execution condition is no longer met after an error end.

*2 If the execution condition is no longer met when an error end occurs, the value of *Error* is TRUE for one task period and it then changes to FALSE.

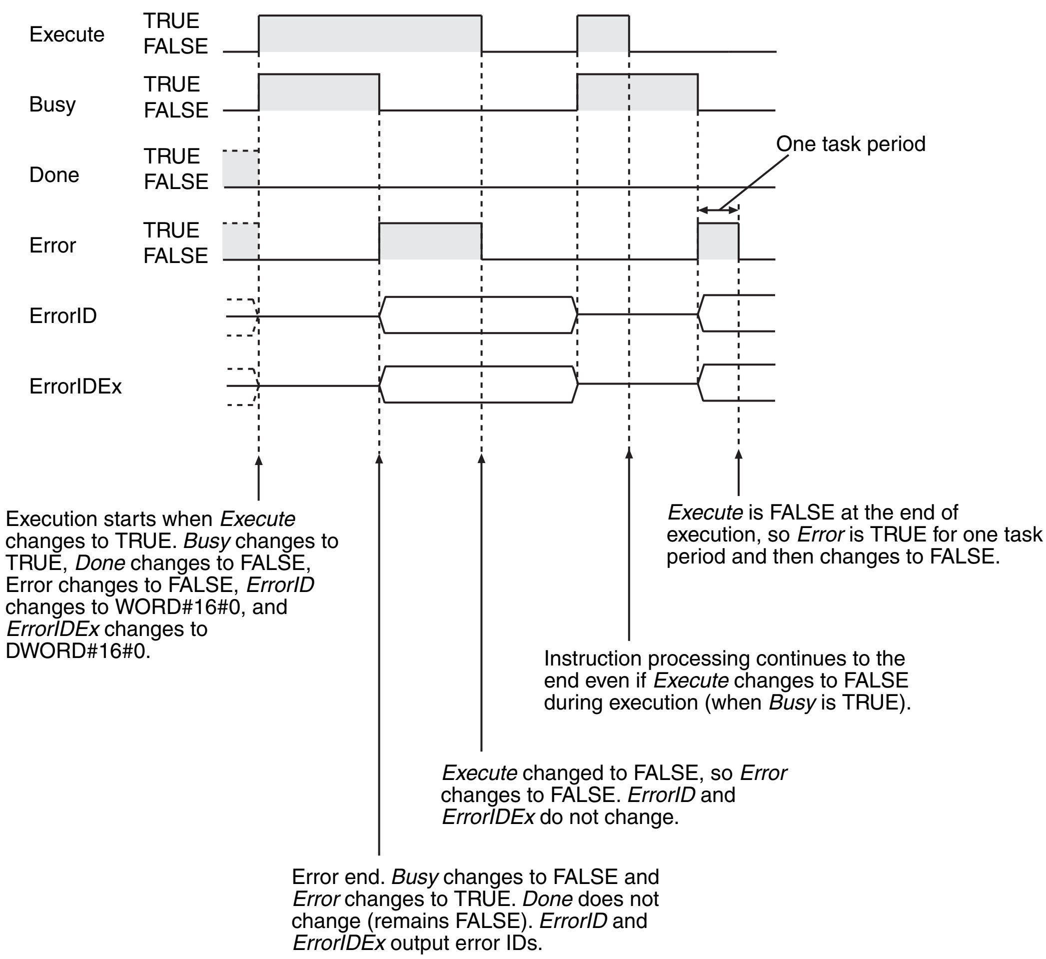
Refer to *A-1 Error Codes That You Can Check with ErrorID* for a list of the error codes that you can check with *ErrorID* and the NY*-series Troubleshooting Manual* (Cat. No. W) for the meanings of the error codes.

Timing charts are provided below for *Execute*, *Done*, *Busy*, *Error*, *ErrorID*, and *ErrorIDEx*.

- **Normal End**



● Error End



Valid Ranges and Default Values of Variables

The valid range of a variable indicates the range of values that variable can take. The default value of a variable indicates the value that is assigned to an input variable when the instruction is executed without a parameter assigned to the input variable. These values are defined for each data type. If specific values are not given for an instruction, then the valid ranges and default values of the data types are applied. These variables are indicated by “depends on data type” in the valid range column and by “---” in the input variable default column. The valid ranges and default values of the data types are given in the following tables.

Classification	Data type	Valid range	Default
Boolean	BOOL	TRUE or FALSE	FALSE
Bit string	BYTE	BYTE#16#00 to FF	BYTE#16#00
	WORD	WORD#16#0000 to FFFF	WORD#16#0000
	DWORD	DWORD#16#00000000 to FFFFFFFF	DWORD#16#0000_0000
	LWORD	LWORD#16#0000000000000000 to FFFFFFFFFFFFFFFF	LWORD#16#0000_0000_0000_0000

Classification	Data type	Valid range	Default
Integers	USINT	USINT#0 to +255	USINT#0
	UINT	UINT#0 to +65535	UINT#0
	UDINT	UDINT#0 to +4294967295	UDINT#0
	ULINT	ULINT#0 to +18446744073709551615	ULINT#0
	SINT	SINT#–128 to +127	SINT#0
	INT	INT#–32768 to +32767	INT#0
	DINT	DINT#–2147483648 to +2147483647	DINT#0
	LINT	LINT#–9223372036854775808 to +9223372036854775807	LINT#0
Real numbers	REAL	REAL#–3.402823e+38 to –1.175495e-38, 0, +1.175495e-38 to +3.402823e+38, +∞/–∞	REAL#0
	LREAL	LREAL#–1.79769313486231e+308 to –2.22507385850721e-308, 0, +2.22507385850721e–308 to +1.79769313486231e+308, +∞/–∞	LREAL#0
Times, durations, dates, and text strings	TIME	T#–9223372036854.775808ms (T#–106751d_23h_47m_16s_854.775808ms) to T#9223372036854.775807ms (T#+106751d_23h_47m_16s_854.775807ms)	T#0s
	DATE	D#1970-01-01 to D#2106-02-06 (January 1, 1970 to February 6, 2106)	D#1970-01-01
	TOD	TOD#00:00:00.000000000 to TOD#23:59:59.999999999 (0:00 and 0.000000000 to 23:59 and 59.999999999 seconds)	TOD#00:00:00.000000000
	DT	DT#1970-01-01-00:00:00.000000000 to DT#2106-02-06-23:59:59.999999999 (0:00 and 0.000000000 on January 1, 1970 to 23:59 and 59.999999999 seconds on February 6, 2106)	DT#1970-01-01-00:00:00.000000000
	STRING	Character code: UTF-8 0 to 1,986 bytes (1,985 single-byte alphanumeric characters plus the final NULL character)	''

Derivative Data Types (Enumerations, Structures, and Unions)

Variables that use derivative data types (enumerations, structures, and unions) are specified as such in the tables of variable data types. The notation is described below.

Enumerations

The data type for an enumerated variable is given within the table. The following is an example. Here, the data type of the *Out* variable is enumerated type _eDAYOFWEEK. The enumerators are described in the description of the function of the instruction.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																	OK		OK	
Out	Refer to *Function* for the enumerators of the enumerated type _eDAYOFWEEK.																			

Structures and Unions

The data type for a structure or union variable is given within the table. The following is an example. Here, the data type of the *In1* variable is structure _sPORT. Details on the members of a structure or union are given in the description of the function of the instruction.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1	Refer to *Function* for details on the structure _sPORT.																			

The tables also indicate any variables for which you can specify a structure, a structure member, a union, or a union member as the parameter.
In the following example, you can specify a parameter with a basic data type, or you can specify a structure, a structure member, a union, or a union member for the *In1* variable. To specify a structure or union, specify only the structure or the union as the parameter. To specify a structure member or a union member, specify the member as the parameter.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	A structure, structure member, union, or union member can also be specified.																			

Array Specifications

Array variable names are followed by “[]” and “(array)” is specified. For these variables, specify an element of the array (i.e., specify the subscript) as the parameter.
An example is shown below. Here, the table shows that *In1[]* is a BYTE array.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[] (array)		OK																		

The data type table indicates the arrays for which structures and unions can be used as elements, as shown in the following example. For these variables, specify an element of the array (i.e., specify the subscript) as the parameter.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[] (array)	Arrays of structures or unions can also be specified.																			

The table indicates any variables for which you can specify either an array or an array element as the parameter.
In the following example, you can specify a parameter with a basic data type, or you can specify an array or an array element. To specify an array, specify only the array as the parameter. To specify an array element, specify an element of the array (i.e., specify the subscript) as the parameter.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1	An array or array element can also be specified.																			

Others

Errors Detected for All Instructions

The errors that can occur for an instruction are given in the *Precautions for Correct Use* section. The following three errors, however, can be detected for any instruction. They are not listed in the *Precautions for Correct Use* sections.

- Reading or writing elements that exceed the range of an array variable.
  Example: Setting *a[4]* for an input variable for the array variable *a[0..3]*.
- Passing parameters that are not variables to instructions for which array variables are defined for input, output, or in-out variables.
- Assigning a text string that is longer than the defined number of bytes to a STRING variable.
- Assigning a text string that does not end in a NULL character to a STRING variable.
- Dividing an integer variable by 0.

Precautions for All Instructions

The amount of processing that is required for some instructions depends on the parameters that you connect. If there is too much processing, the instruction execution time increases and the task period may be exceeded. This will result in a Task Period Exceeded error. Adjust the amount of processing to a suitable amount.

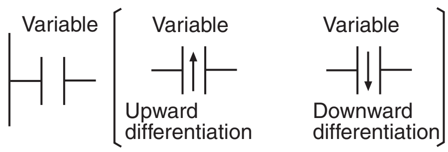
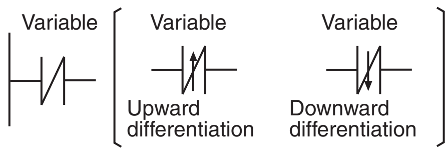
Ladder Diagram Instructions

Instruction	Name	Page
LD and LDN	Load/ Load NOT	2-16
AND and ANDN	AND/ AND NOT	2-18
OR and ORN	OR/ OR NOT	2-20
Out and OutNot	Output/ Output NOT	2-22

LD and LDN

LD: Reads the value of a BOOL variable.

LDN: Reads the inverse of the value of a BOOL variable.

Instruction	Name	FB/FUN	Graphic expression	ST expression
LD	Load	---		None
LDN	Load NOT	---		None

Variables

None

Function

● LD

The LD instruction reads the value of the specified BOOL variable and outputs it to the next instruction. If the value of the specified variable is TRUE, then TRUE is output. If the value is FALSE, then FALSE is output. Use the LD instruction for the first NO bit from the bus bar or for the first NO bit of a logic block.

● LDN

The LD instruction reads the inverse of the value of the specified BOOL variable and outputs it to the next instruction. If the value of the specified variable is TRUE, then FALSE is output. If the value is FALSE, then TRUE is output. Use the LDN instruction for the first NC bit from the bus bar or for the first NC bit of a logic block.

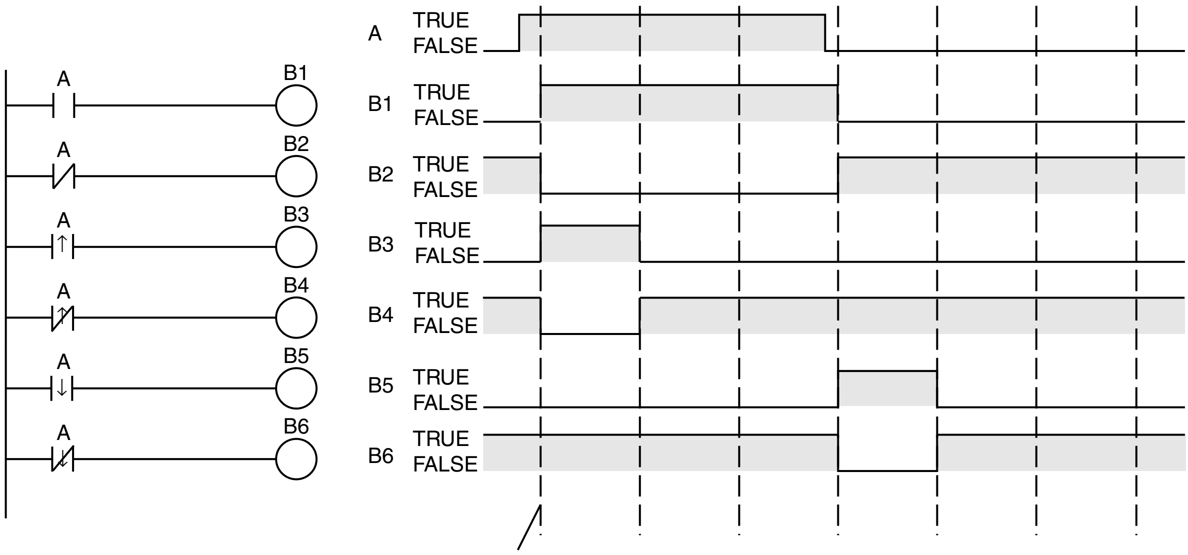
The operation is as shown below if you do not specify upward or downward differentiation.

Instruction	Value of variable	Output value
LD	TRUE	TRUE
	FALSE	FALSE
LDN	TRUE	FALSE
	FALSE	TRUE

If you specify upward or downward differentiation, the operation depends on the following: the value of the variable the last time the instruction was executed and the current value of the variable. This is shown below.

Instruction	Differentiation specification	Value of variable at last execution and current value of variable	Output value
LD	Upward differentiation	FALSE at the last execution → Currently TRUE	TRUE
		Other than the above.	FALSE
	Downward differentiation	TRUE at the last execution → Currently FALSE	TRUE
		Other than the above.	FALSE
LDN	Upward differentiation	FALSE at the last execution → Currently TRUE	FALSE
		Other than the above.	TRUE
	Downward differentiation	TRUE at the last execution → Currently FALSE	FALSE
		Other than the above.	TRUE

The following figure shows a programming example and timing chart.



Precautions for Correct Use

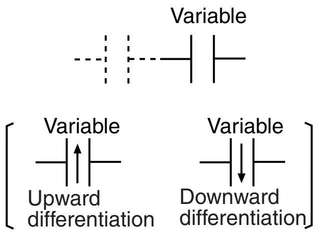
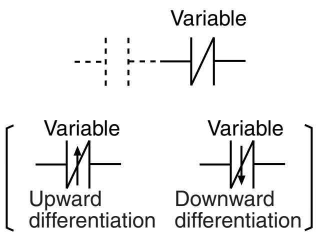
- An error occurs in the following case and the output value from the last execution is retained.
  - You specify an array element for the variable value and the element does not exist.

    Example: A BOOL array a[0..5] is defined, but the instruction is executed using a[10] as the variable.
- Do not use these instructions as the rightmost instruction on a rung. If you do, an error occurs on the Sysmac Studio and you cannot transfer the user program to the Controller.

AND and ANDN

AND: Takes the logical AND of the value of a BOOL variable and the execution condition.

ANDN: Takes the logical AND of the inverse of the value of a BOOL variable and the execution condition.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AND	AND	---		result:=vBool1 AND vBOOL2; result:=vBool1 & vBool2;
ANDN	AND NOT	---		result:=vBool1 AND NOT vBool2;

Variables

None

Function

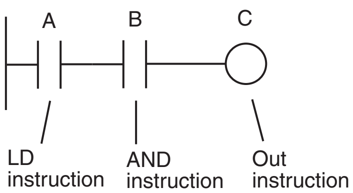
● AND

The AND instruction takes the logical AND of the value of a specified BOOL variable and the execution condition and outputs it to the next instruction. Use the AND instruction for a NO bit connected in series with the previous instruction.

● ANDN

The ANDN instruction takes the logical AND of the inverse of the value of a specified BOOL variable and the execution condition and outputs it to the next instruction. Use the ANDN instruction for a NC bit connected in series with the previous instruction.

The following figure shows a programming example of the AND instruction. It takes the logical AND of variable *A* and variable *B* and outputs it to variable *C*.



It takes the logical AND of variable *A* and variable *B* and outputs the result to variable *C*.

The operation is as shown below if you do not specify upward or downward differentiation.

Instruction	Combination of variable value and execution condition	Output value
AND	Variable value: TRUE Execution condition: TRUE	TRUE
	Other than the above.	FALSE
ANDN	Variable value: FALSE Execution condition: TRUE	TRUE
	Other than the above.	FALSE

If you specify upward or downward differentiation, the operation depends on the following: the value of the variable the last time the instruction was executed, the current value of the variable, and the execution condition. This is shown below.

Instruction	Differentiation specification	Combination of value of variable at last execution, current value of variable, and execution condition	Output value
AND	Upward differentiation	Variable value: FALSE at the last execution → Currently TRUE Execution condition: TRUE	TRUE
		Other than the above.	FALSE
	Downward differentiation	Variable value: TRUE at the last execution → Currently FALSE Execution condition: TRUE	TRUE
		Other than the above.	FALSE
ANDN	Upward differentiation	Variable value: FALSE at the last execution → Currently TRUE Execution condition: TRUE	FALSE
		Variable value: Ignored Execution condition: FALSE	
		Other than the above.	TRUE
	Downward differentiation	Variable value: TRUE at the last execution → Currently FALSE Execution condition: TRUE	FALSE
		Variable value: Ignored Execution condition: FALSE	
		Other than the above.	TRUE

Precautions for Correct Use

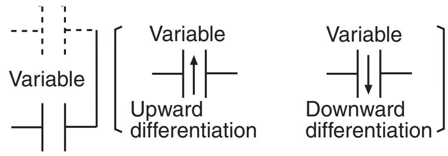
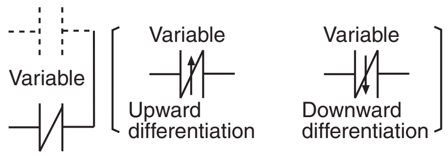
- An error occurs in the following case and the output value from the last execution is retained.
  - You specify an array element for the variable value and the element does not exist.

    Example: A BOOL array a[0..5] is defined, but the instruction is executed using a[10] as the variable.
- Do not use these instructions as the rightmost instruction on a rung. If you do, an error occurs on the Sysmac Studio and you cannot transfer the user program to the Controller.
- You cannot connect these instructions directly to the bus bar.

OR and ORN

OR: Takes the logical OR of the value of a BOOL variable and the execution condition.

ORN: Takes the logical OR of the inverse of the value of a BOOL variable and the execution condition.

Instruction	Name	FB/FUN	Graphic expression	ST expression
OR	OR	---		result:=vBool1 OR vBool2;
ORN	OR NOT	---		result:=vBool1 OR NOT vBool2;

Variables

None

Function

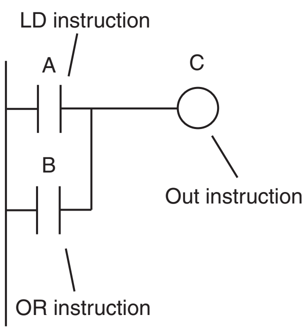
● OR

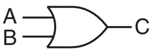
The OR instruction takes the logical OR of the value of a specified BOOL variable and the execution condition and outputs it to the next instruction. Use the OR instruction for a NO bit connected in parallel with the previous instruction. Use the OR instruction to configure a logical OR between an NO bit and one of the following: a LD or LDN instruction connected directly to the bus bar, or the logic block starting with a LD or LDN instruction and ending with the instruction immediately before the OR instruction.

● ORN

The ORN instruction takes the logical OR of the inverse of the value of a specified BOOL variable and the execution condition and outputs it to the next instruction. Use the ORN instruction for a NC bit connected in parallel with the previous instruction. Use the ORN instruction to configure a logical OR between an NC bit and one of the following: a LD or LDN instruction connected directly to the bus bar, or the logic block starting with a LD or LDN instruction and ending with the instruction immediately before the ORN instruction.

The following figure shows a programming example of the OR instruction. It takes the logical OR of variable *A* and variable *B* and outputs it to variable *C*.





It takes the logical OR of variable *A* and variable *B* and outputs the result to variable *C*.

The operation is as shown below if you do not specify upward or downward differentiation.

Instruction	Combination of variable value and execution condition	Output value
OR	Variable value: FALSE Execution condition: FALSE	FALSE
	Other than the above.	TRUE
ORN	Variable value: TRUE Execution condition: FALSE	FALSE
	Other than the above.	TRUE

If you specify upward or downward differentiation, the operation depends on the following: the value of the variable the last time the instruction was executed, the current value of the variable, and the execution condition. This is shown below.

Instruction	Differentiation specification	Combination of value of variable at last execution, current value of variable, and execution condition	Output value
OR	Upward differentiation	Variable value: FALSE at the last execution → Currently TRUE Execution condition: Ignored.	TRUE
		Variable value: Ignored Execution condition: TRUE	
		Other than the above.	FALSE
	Downward differentiation	Variable value: TRUE at the last execution → Currently FALSE Execution condition: Ignored.	TRUE
		Variable value: Ignored Execution condition: TRUE	
		Other than the above.	FALSE
ORN	Upward differentiation	Variable value: FALSE at the last execution → Currently TRUE Execution condition: FALSE	FALSE
		Other than the above.	TRUE
	Downward differentiation	Variable value: TRUE at the last execution → Currently FALSE Execution condition: FALSE	FALSE
		Other than the above.	TRUE

Precautions for Correct Use

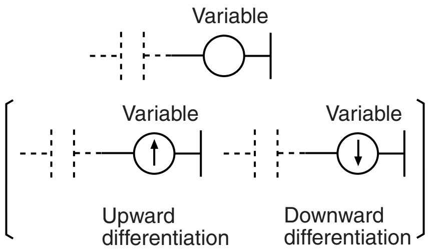
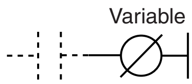
- An error occurs in the following case and the output value from the last execution is retained.
  - You specify an array element for the variable value and the element does not exist.

    Example: A BOOL array a[0..5] is defined, but the instruction is executed using a[10] as the variable.
- Do not use these instructions as the rightmost instruction on a rung. If you do, an error occurs on the Sysmac Studio and you cannot transfer the user program to the Controller.

Out and OutNot

Out: Takes the logical result from the previous instruction and outputs it to a BOOL variable.

OutNot: Takes the inverse of the logical result from the previous instruction and outputs it to a BOOL variable.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Out	Output	---		Variable:=(Logic expression up to previous instruction);
OutNot	Output NOT	---		Variable:=NOT(Logic expression up to previous instruction);

Variables

None

Function

● Out

The Out instruction takes the logical result from the previous instruction and outputs it to a specified BOOL variable.

The operation is as shown below if you do not specify upward or downward differentiation.

Logic processing result from previous instruction	Output
TRUE	TRUE
FALSE	FALSE

You can specify upward or downward differentiation for the Out instruction. If upward or downward differentiation is specified, the output value is determined by changes in the result of logic processing from the previous instruction between the last execution of the instruction and the current execution. The operation is according to the current logical result from the previous instruction, as shown in the following table.

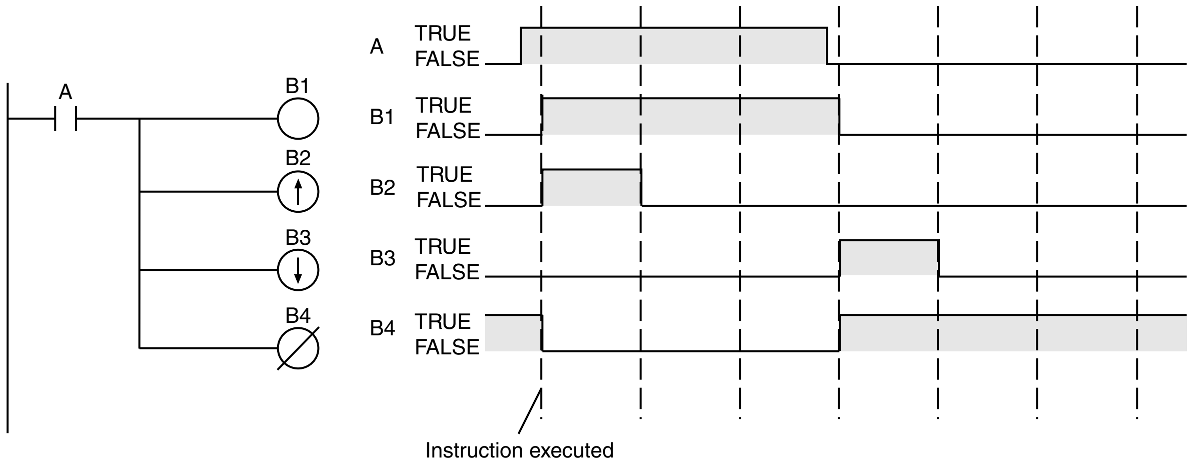
Differentiation specification	Results of logic processing from the previous execution and current execution	Output
Upward differentiation	FALSE at the last execution → Currently TRUE	TRUE
	Other than the above.	FALSE
Downward differentiation	TRUE at the last execution → Currently FALSE	TRUE
	Other than the above.	FALSE

● OutNot

The OutNot instruction takes the inverse of the logical result from the previous instruction and outputs it to a specified BOOL variable.

Logic processing result from previous instruction	Output
TRUE	FALSE
FALSE	TRUE

The following figure shows a programming example and timing chart.



Additional Information

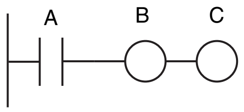
Differences between the Set and Reset Instructions and the Out and OutNot Instructions

- The Set and Reset instructions operate only when the input value changes to TRUE. They do not operate when the input value is FALSE. When the input value is FALSE, the output does not change.
- The Out and OutNot instructions affect the output whether the logical result of the previous instruction is TRUE or FALSE.

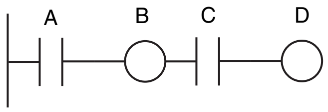
Precautions for Correct Use

- In the following case, an error occurs and nothing is output.
  - You specify an array element for the variable value and the element does not exist.

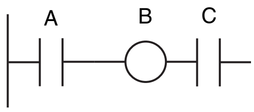
    Example: A BOOL array a[0..5] is defined, but the instruction is executed using a[10] as the variable.
- The following connections are possible.
  - You can connect another Out instruction after an Out instruction.



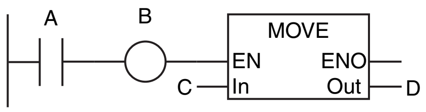
  - You can connect an LD instruction and Out instruction after an Out instruction.



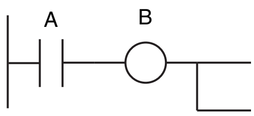
- The following connections are not possible.
  - You cannot connect only an LD instruction after an Out instruction.

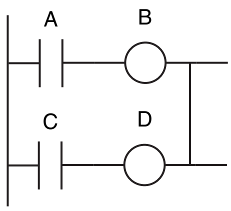


  - Functions and function blocks cannot be connected after an Out instruction.



  - Branches and joins cannot be used after Out instructions.





ST Statement Instructions

IF

The IF construct uses the evaluation result of a specified condition expression to select one of two statements to execute.

Instruction	Name	FB/FUN	Graphic expression	ST expression

IF	If	---	None	IF *condition expression* THEN
				   statement;
				ELSIF *condition expression* THEN
				   statement;
				ELSE
				   statement;
				END_IF;

Variables

None

Function

The IF construct uses the evaluation result of a specified condition expression to select one of two statements to execute. Use a condition expression that evaluates to TRUE or FALSE.

Item used for condition expression	Example	Evaluation result
Logic expression	a>3	If the value of variable *a* is greater than 3, the result is TRUE. Otherwise, the result is FALSE.
	a=b	If the values of variables *a* and *b* are equal, the result is TRUE. Otherwise, the result is FALSE.
BOOL variable	abc	If the value of variable *abc* is TRUE, the result is TRUE. If it is FALSE, the result is FALSE.
BOOL constant	TRUE	TRUE
Function with a BOOL return value	FUN name	If the function returns TRUE, the result is TRUE. If it returns FALSE, the result is FALSE.

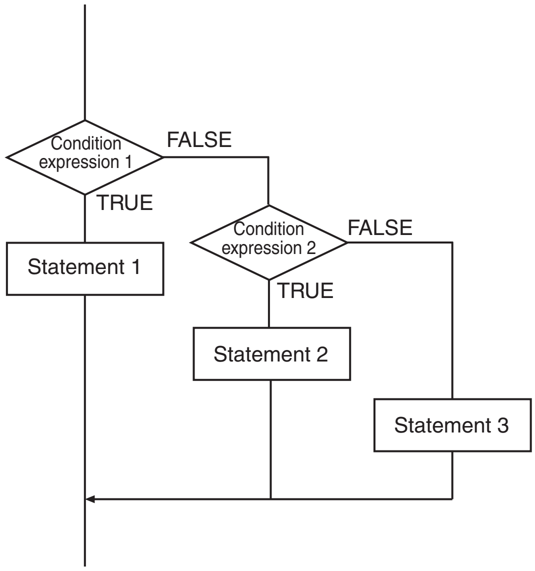
You can use the following operators in the logic expression.

Operator	Meaning	Example	Evaluation result
=	Equals	a=b	If the values of variables *a* and *b* are equal, the result is TRUE. Otherwise, the result is FALSE.
<>	Not equals	a<>b	If the values of variables *a* and *b* are not equal, the result is TRUE. Otherwise, the result is FALSE.
<	Comparison	a<b	If the value of variable *a* is less than the value of variable *b*, the result is TRUE. Otherwise, the result is FALSE.
<=		a<=b	If the value of variable *a* is less than or equal to the value of variable *b*, the result is TRUE. Otherwise, the result is FALSE.
>		a>b	If the value of variable *a* is greater than the value of variable *b*, the result is TRUE. Otherwise, the result is FALSE.
>=		a>=b	If the value of variable *a* is greater than or equal to the value of variable *b*, the result is TRUE. Otherwise, the result is FALSE.
AND (&)	Logical AND	a AND b a & b	The result is the logical AND of BOOL variables *a* and *b*.

Operator	Meaning	Example	Evaluation result
OR	Logical OR	a OR b	The result is the logical OR of BOOL variables *a* and *b*.
XOR	Exclusive OR	a XOR b	The result is the logical exclusive OR of BOOL variables *a* and *b*.
NOT	NOT	NOT a	The result is the NOT of BOOL variable *a*.

The flowchart in the following example shows the evaluation results for condition expressions 1 and 2. You can use more than one statement for each of statements 1 to 3.

IF *condition expression 1* THEN
    statement 1;
ELSIF *condition expression 2* THEN
    statement 2;
ELSE
    statement 3;
END_IF;



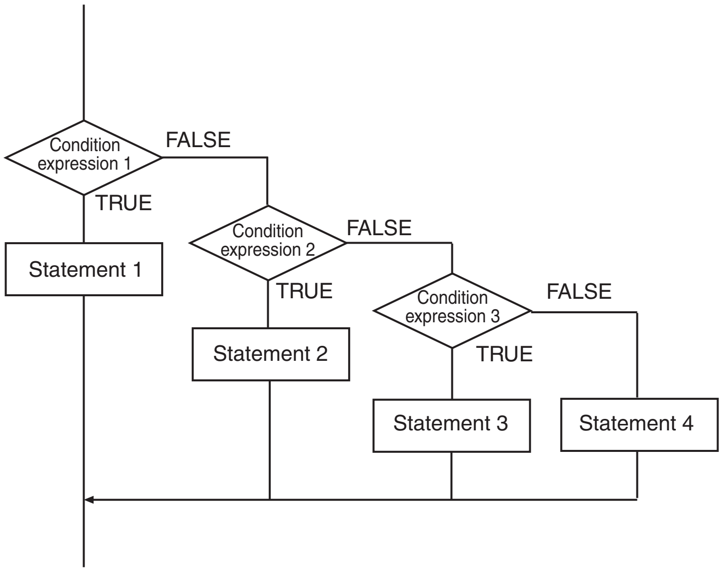
Additional Information

- You can use the IF construct to build a hierarchy. The following example executes statement 11 if the evaluation results of both condition expression 1 and condition expression 11 are TRUE.

IF *condition expression 1* THEN
    IF *condition expression 11* THEN
        statement 11;
    ELSIF *condition expression 12* THEN
        statement 12;
    ELSE
        statement 13;
    END_IF;
ELSIF *condition expression 2* THEN
    statement 2;
ELSE
    statement 3;
END_IF;

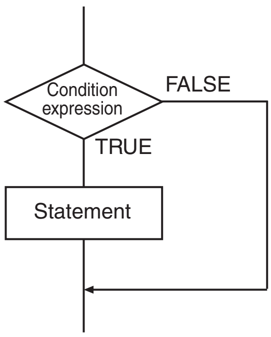
You can use ELSIF more than once. The following processing flow is for this example.

IF *condition expression 1* THEN
    statement 1;
ELSIF *condition expression 2* THEN
    statement 2;
ELSIF *condition expression 3* THEN
    statement 3;
ELSE
    statement 4;
END_IF;



- You do not use ELSIF if there is only one condition expression. You do not use ELSE if no processing is performed when none of the condition expressions are TRUE. The following processing flow is for this example.

IF condition expression THEN
    statement;
END_IF;



- There are no restrictions on the statements that you can use. You can use the same types of statements for the statements in the IF construct as you do for the statements outside the IF construct. For example, you can use function block calls and FOR constructs.

Precautions for Correct Use

- You must always use IF and END_IF. They must be paired.
- You can use a hierarchy that is 15 levels deep, but count all levels of IF, CASE, FOR, WHILE, and REPEAT constructs.

Sample Programming

This example assigns INT#0 to variable *def* if the value of variable *abc* is less than INT#0. It assigns INT#1 to variable *def* and INT#2 to variable *ghi* if the value of variable *abc* is INT#0. It assigns INT#3 to variable *def* if the value of variable *abc* is none of the above.

Variable	Data type	Initial value
abc	INT	0
def	INT	0
ghi	INT	0

```
IF (abc<INT#0) THEN
    def:=INT#0;
ELSIF (abc=INT#0) THEN
    def:=INT#1;
    ghi:=INT#2;
ELSE
    def:=INT#3;
END_IF;
```

CASE

You use the CASE construct to select the statement to execute based on the value of a specified integer expression.

Instruction	Name	FB/FUN	Graphic expression	ST expression
CASE	Case	---	None	CASE *integer expression* OF *value*: *statement*; *value*: *statement*; . . . ELSE *statement*; END_CASE;

Variables

None

Function

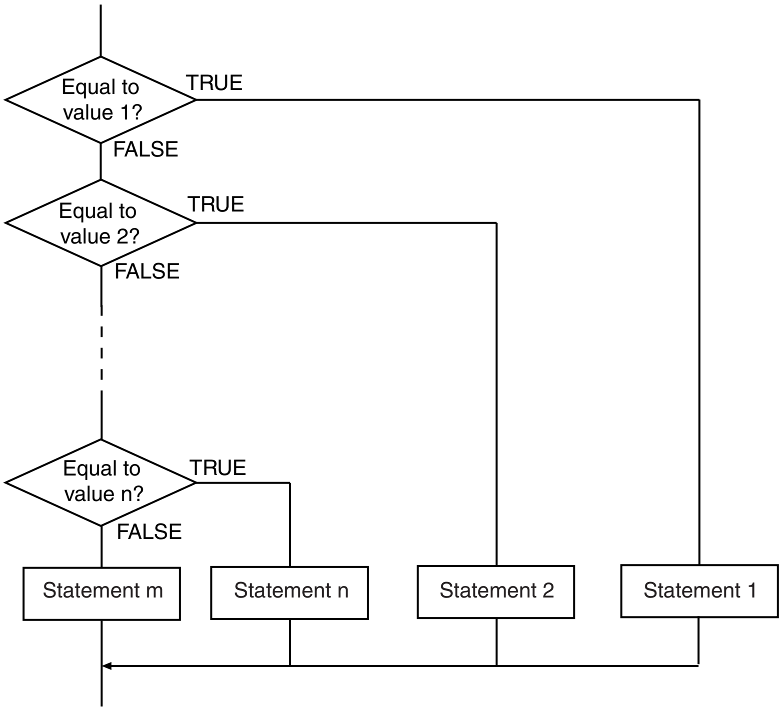
You use the CASE construct to select the statement to execute based on the value of a specified integer expression.

You can use any of the following as the integer expression and values.

	Allowed notation
Integer expression	Integer variable, integer constant, integer expression, or a function that returns an integer return value, enumeration variable, enumeration expression, or enumerator
Values	Integer constants

The flowchart in the following example shows the processing flow for an integer expression. You can use more than one statement for each of the statements.

CASE *integer expression* OF
  1 :
    statement 1;
  2 :
    statement 2;
  :
  :
  n :
    statement n;
  ELSE
    statement m;
END_CASE;



Additional Information

- You can use the CASE construct to build a hierarchy. The following example executes statement 12 if the value of integer expression 1 is 1 and the value of integer expression 11 is 2.

```
CASE integer expression 1 OF
    1 :
       CASE integer expression 1 OF
       1 :
          statement 11;
       2 :
          statement 12;
       ELSE
          statement 1m;
       END_CASE;
    2 :
       statement 2;
    3 :
       statement 3;
    ELSE
       statement m;
END_CASE;
```

- You can use more than one value at the same time. Separate values with commas. The following example executes statement 1 if the value of the integer expression is either 1 or 2.

```
CASE integer expression 1 OF
    1,2 :
       statement 1;
    3 :
       statement 2;
    4 :
       statement 3;
    ELSE
       statement m;
END_CASE;
```

- You can use a range of consecutive values. Place two periods between the numbers to indicate consecutive values. The following example executes statement 1 if the value of the integer expression is between 10 and 15, inclusive.

```
CASE integer expression 1 OF
    10..15:
        statement 1;
    16:
        statement 2;
    17:
        statement 3;
    ELSE
        statement m;
END_CASE;
```

- You can omit ELSE. If you do, none of the statements is executed if none of the values is equal to the value of the integer expression.
- There are no restrictions on the statements that you can use. You can use the same types of statements for the statements in the CASE construct as you do for the statements outside the CASE construct. For example, you can use function block calls and FOR constructs.
- The following is different in comparison to a C language *switch* statement. With a C language *switch* statement, all statements after a value that equals the integer expression are executed unless a *break* statement is used. With the CASE statement, only the statements that correspond directly to the value that equals the integer expression are executed. For example, in the following example, statements 1 to 3 are executed for the C language *switch* statement. Here, only statement 1 is executed for the CASE instruction.

C Language switch Statement

```
val=1;
switch val
{
        case 1:
    statement 1;
        case 2:
    statement 2;
        case 3:
    statement 3;
}
```

CASE Instruction

```
val:=1;
CASE val OF
    1:
        statement 1;
    2:
        statement 2;
    3:
        statement 3;
END_CASE;
```

Precautions for Correct Use

- You must always use CASE and END_CASE. They must be paired.
- The data types of the integer expression and values can be different.
- Each value can be given only once.
- You can use a hierarchy that is 15 levels deep, but count all levels of IF, CASE, FOR, WHILE, and REPEAT constructs.

Sample Programming

This example assigns INT#10 to variable *def* if the value of variable *abc* is INT#1, INT#20 if the value of variable *abc* is INT#2, and INT#30 if the value of variable *abc* is INT#3. Otherwise, it assigns the value of variable *ghi* to variable *def*.

Variable	Data type	Initial value
abc	INT	0
def	INT	0
ghi	INT	0

```
CASE abc OF
  INT#1:
     def:=INT#10;
  INT#2:
     def:=INT#20;
  INT#3:
     def:=INT#30;
  ELSE
     def:=ghi;
END_CASE;
```

This example assigns INT#10 to variable *def* if the value of variable *abc* is INT#1, INT#20 if the value of variable *abc* is INT#2 or INT#5, and INT#30 if the value of variable *abc* is between INT#6 and INT#10, inclusive. Otherwise, it does nothing.

Variable	Data type	Initial value
abc	INT	0
def	INT	0

```
CASE abc OF
  INT#1:
     def:=INT#10;
  INT#2,INT#5:
     def:=INT#20;
  INT#6..INT#10:
     def:=INT#30;
END_CASE;
```

WHILE

The WHILE construct repeatedly executes a statement as long as the evaluation result of a specified condition expression is TRUE.

Instruction	Name	FB/FUN	Graphic expression	ST expression
WHILE	While	---	None	WHILE *condition expression* DO *statement*; END_WHILE;

Variables

None

Function

The WHILE construct repeatedly executes a statement as long as the evaluation result of a specified condition expression is TRUE. Use a condition expression that evaluates to TRUE or FALSE.

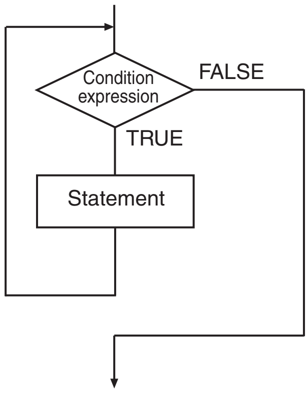
Item used for condition expression	Example	Evaluation result
Logic expression	a>3	If the value of variable *a* is greater than 3, the result is TRUE. Otherwise, the result is FALSE.
	a=b	If the values of variables *a* and *b* are equal, the result is TRUE. Otherwise, the result is FALSE.
BOOL variable	abc	If the value of variable *abc* is TRUE, the result is TRUE. If it is FALSE, the result is FALSE.
BOOL constant	TRUE	TRUE
Function with a BOOL return value	FUN name	If the function returns TRUE, the result is TRUE. If it returns FALSE, the result is FALSE.

You can use the following operators in the logic expression.

Operator	Meaning	Example	Evaluation result
=	Equals	a=b	If the values of variables *a* and *b* are equal, the result is TRUE. Otherwise, the result is FALSE.
<>	Not equals	a<>b	If the values of variables *a* and *b* are not equal, the result is TRUE. Otherwise, the result is FALSE.
<	Comparison	a<b	If the value of variable *a* is less than the value of variable *b*, the result is TRUE. Otherwise, the result is FALSE.
<=		a<=b	If the value of variable *a* is less than or equal to the value of variable *b*, the result is TRUE. Otherwise, the result is FALSE.
>		a>b	If the value of variable *a* is greater than the value of variable *b*, the result is TRUE. Otherwise, the result is FALSE.
>=		a>=b	If the value of variable *a* is greater than or equal to the value of variable *b*, the result is TRUE. Otherwise, the result is FALSE.
AND (&)	Logical AND	a AND b a & b	The result is the logical AND of BOOL variables *a* and *b.*
OR	Logical OR	a OR b	The result is the logical OR of BOOL variables *a* and *b*.
XOR	Exclusive OR	a XOR b	The result is the logical exclusive OR of BOOL variables *a* and *b*.
NOT	NOT	NOT a	The result is the NOT of BOOL variable *a*.

The following processing flow is for this example. You can use more than one statement.

WHILE *condition expression* DO
    statement;
END_WHILE;



Additional Information

- The statement is not executed even once if the condition expression is FALSE the first time it is evaluated.
- There are no restrictions on the statements that you can use. You can use the same types of statements for the statements in the WHILE construct as you do for the statements outside the WHILE construct. For example, you can use function block calls and FOR constructs.

Precautions for Correct Use

- You must always use WHILE and END_WHILE. They must be paired.
- You can use a hierarchy that is 15 levels deep, but count all levels of IF, CASE, FOR, WHILE, and REPEAT constructs.

Sample Programming

This example adds INT#7 to variable *abc* as long as the value of variable *abc* is less than or equal to INT#1000.

Variable	Data type	Initial value
abc	INT	0

```
abc:=INT#0;
WHILE abc<=INT#1000 DO
  abc:=abc+INT#7;
END_WHILE;
```

REPEAT

The REPEAT construct executes a statement once and then executes it repeatedly until a specified condition expression is TRUE.

Instruction	Name	FB/FUN	Graphic expression	ST expression
REPEAT	Repeat	---	None	REPEAT *statement*; UNTIL condition expression END_REPEAT;

Variables

None

Function

The REPEAT construct executes a statement once and then executes it repeatedly until a specified condition expression is TRUE. Use a condition expression that evaluates to TRUE or FALSE.

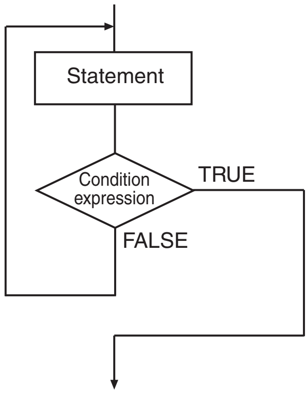
Item used for condition expression	Example	Evaluation result
Logic expression	a>3	If the value of variable *a* is greater than 3, the result is TRUE. Otherwise, the result is FALSE.
	a=b	If the values of variables *a* and *b* are equal, the result is TRUE. Otherwise, the result is FALSE.
BOOL variable	abc	If the value of variable *abc* is TRUE, the result is TRUE. If it is FALSE, the result is FALSE.
BOOL constant	TRUE	TRUE
Function with a BOOL return value	FUN name	If the function returns TRUE, the result is TRUE. If it returns FALSE, the result is FALSE.

You can use the following operators in the logic expression.

Operator	Meaning	Example	Evaluation result
=	Equals	a=b	If the values of variables *a* and *b* are equal, the result is TRUE. Otherwise, the result is FALSE.
<>	Not equals	a<>b	If the values of variables *a* and *b* are not equal, the result is TRUE. Otherwise, the result is FALSE.
<	Comparison	a<b	If the value of variable *a* is less than the value of variable *b*, the result is TRUE. Otherwise, the result is FALSE.
<=		a<=b	If the value of variable *a* is less than or equal to the value of variable *b*, the result is TRUE. Otherwise, the result is FALSE.
>		a>b	If the value of variable *a* is greater than the value of variable *b*, the result is TRUE. Otherwise, the result is FALSE.
>=		a>=b	If the value of variable *a* is greater than or equal to the value of variable *b*, the result is TRUE. Otherwise, the result is FALSE.
AND (&)	Logical AND	a AND b a & b	The result is the logical AND of BOOL variables *a* and *b*.
OR	Logical OR	a OR b	The result is the logical OR of BOOL variables *a* and *b*.
XOR	Exclusive OR	a XOR b	The result is the logical exclusive OR of BOOL variables *a* and *b*.
NOT	NOT	NOT a	The result is the NOT of BOOL variable *a*.

The following processing flow is for this example. You can use more than one statement.

REPEAT
    statement;
UNTIL *condition expression*
END_REPEAT;



Additional Information

- The statement is executed once before the condition expression is evaluated. Therefore, the statement is always executed at least once.
- There are no restrictions on the statements that you can use. You can use the same types of statements for the statements in the REPEAT construct as you do for the statements outside the REPEAT construct. For example, you can use function block calls and FOR constructs.

Precautions for Correct Use

- You must always use REPEAT, UNTIL, and END_REPEAT. They must be used as a set.
- You can use a hierarchy that is 15 levels deep, but count all levels of IF, CASE, FOR, WHILE, and REPEAT constructs.

Sample Programming

This example adds INT#1 to variable *abc* until the value of variable *abc* exceeds INT#10.

Variable	Data type	Initial value
abc	INT	0

```
abc:=INT#0;
REPEAT
  abc:=abc+INT#1;
UNTIL abc>INT#10
END_REPEAT;
```

EXIT

The EXIT instruction is used to end repeat processing from the lowest level FOR, WHILE, or REPEAT instruction.

Instruction	Name	FB/FUN	Graphic expression	ST expression

EXIT	Break Loop	---	None	FOR Index:=0 TO 9 BY 1 DO
				  IF Error[Index] THEN
				    EXIT;
				  END_IF;
				END_FOR;

Variables

None

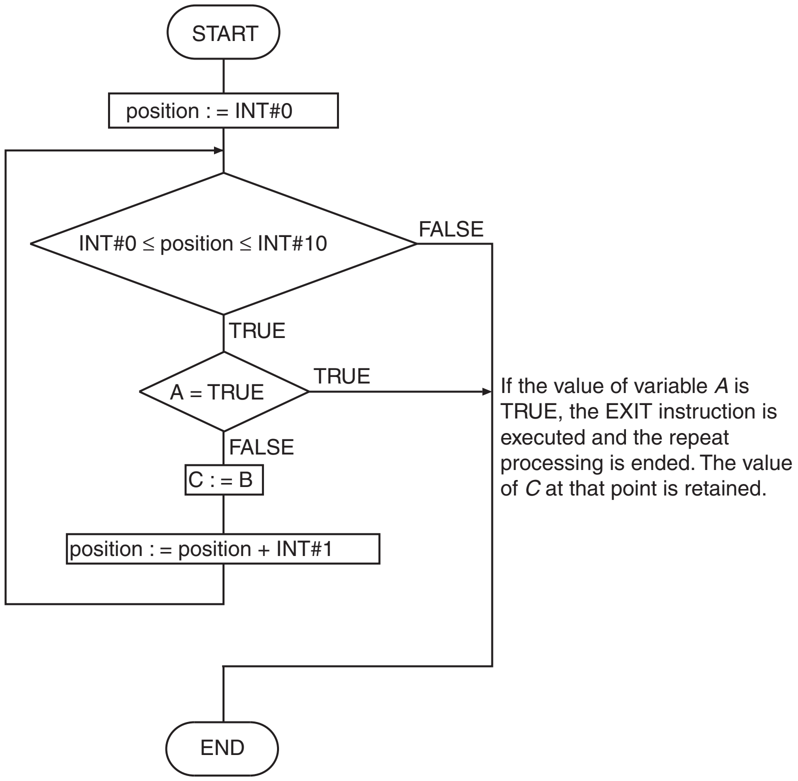
Function

The EXIT instruction is used to end repeat processing from the lowest level FOR, WHILE, or REPEAT instruction. Processing moves to the next instruction after the repeat processing.

In the following programming, the value of variable *A* is checked every repetition during repeat processing for the FOR instruction. If the value of variable *A* is TRUE, the EXIT instruction is executed and the repeat processing is ended. If that occurs, *C:=B;* following END_IF is not executed and the value of variable *C* is retained.

```
FOR position:=INT#0 TO INT#10 BY INT#1 DO
     IF (A=TRUE) THEN
        EXIT;
    END_IF;
    C:=B;
END_FOR;
```

The flowchart for this programming is given below.

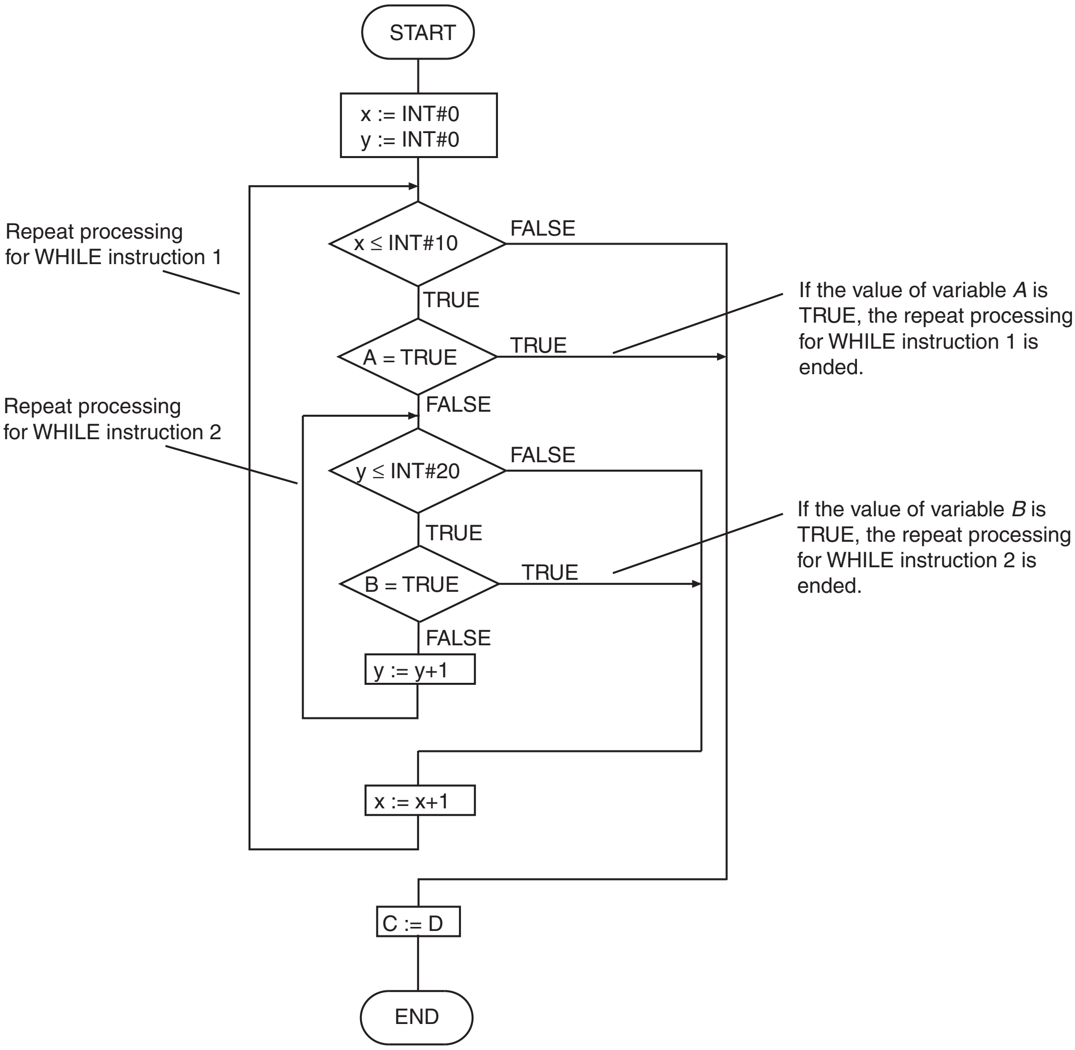


When the EXIT instruction is executed, only the lowest level of repeat processing is ended. Therefore, in the following programming, when the value of variable *B* is TRUE, EXIT instruction 2 is executed and the repeat processing for WHILE instruction 2 is ended. As the result, processing moves to *x:=x+1;*. In this case, repeat processing for WHILE instruction 1 (one level higher) is continued.

If the value of variable *A* is TRUE, EXIT instruction 1 is executed and the repeat processing for WHILE instruction 1 is ended. As the result, processing moves to *C:=D;*.

```
x:=INT#0;
y:=INT#0;
WHILE x<=INT#10 DO        // WHILE instruction 1
    IF (A=TRUE) THEN
        EXIT;             // EXIT instruction 1
    END_IF;
    WHILE y<=INT#20 DO    // WHILE instruction 2
        IF (B=TRUE) THEN
            EXIT;         // EXIT instruction 2
        END_IF;
        y := y+1;
    END_WHILE;
    x = x+1;
END_WHILE
C:=D;
```

The flowchart for this programming is given below.



Precautions for Correct Use

- Always place this instruction between the FOR and END_FOR, WHILE and END_WHILE, or REPEAT and END_REPEAT instructions.
- If you nest repeat processing, one EXIT instruction is required for each nesting level to end all of the repeat processing.

RETURN

Refer to *RETURN* on page *2-67* in the Sequence Control Instructions for a description of this instruction.

FOR

Refer to *FOR and NEXT* on page *2-82* in the Sequence Control Instructions for a description of this instruction.

Sequence Input Instructions

Instruction	Name	Page
R_TRIG (Up) and F_TRIG (Down)	Up Trigger/ Down Trigger	2-44
TestABit and TestABitN	Test A Bit/ Test A Bit NOT	2-47

R_TRIG (Up) and F_TRIG (Down)

R_TRIG (Up): Outputs TRUE for one task period only when the input signal changes to TRUE.
F_TRIG (Down): Outputs TRUE for one task period only when the input signal changes to FALSE.

Instruction	Name	FB/FUN	Graphic expression	ST expression
R_TRIG	Up Trigger	FB	R_TRIG_instance R_TRIG Clk Q	R_TRIG_instance(Clk, Q);
Up		FUN	Up In Out	None
F_TRIG	Down Trigger	FB	F_TRIG_instance F_TRIG Clk Q	F_TRIG_instance(Clk, Q);
Down		FUN	Down In Out	None

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Clk, In	Input signal	Input	Input signal	Depends on data type.	---	---
Q, Out	Output signal	Output	Output signal	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Clk, In	OK																			
Q, Out	OK																			

Function

● R_TRIG

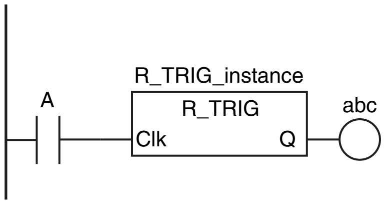
R_TRIG assigns TRUE to output signal *Q* for one task period only when input signal *Clk* changes to TRUE. Otherwise, the value of *Q* is FALSE.

● Up

The functions of the R_TRIG instruction and the Up instruction are the same. The *Clk* variable of the F_TRIG instruction corresponds to the *In* variable of the Down instruction. The *Q* variable corresponds to the *Out* variable. However, the operation of the Up instruction is different from the operation of the R_TRIG instruction in the first task period in which it is executed. Refer to the *Precautions for Correct Use* for the operation of the Up instruction in the first task period in which it is executed.

The following figure shows a programming example and timing chart.

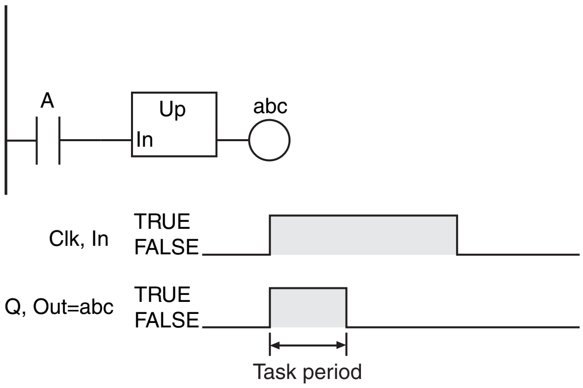
LD



ST

```
R_TRIG_instance(A, abc);
```

LD



● F_TRIG

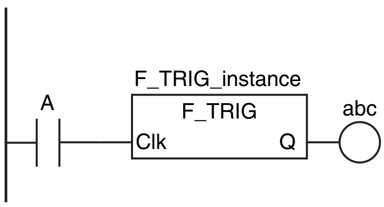
F_TRIG assigns TRUE to output signal *Q* for one task period only when input signal *Clk* changes to FALSE. Otherwise, the value of *Q* is FALSE.

● Down

The functions of the F_TRIG instruction and the Down instruction are exactly the same. The *Clk* variable of the F_TRIG instruction corresponds to the *In* variable of the Down instruction. The *Q* variable corresponds to the *Out* variable.

The following figure shows a programming example and timing chart.

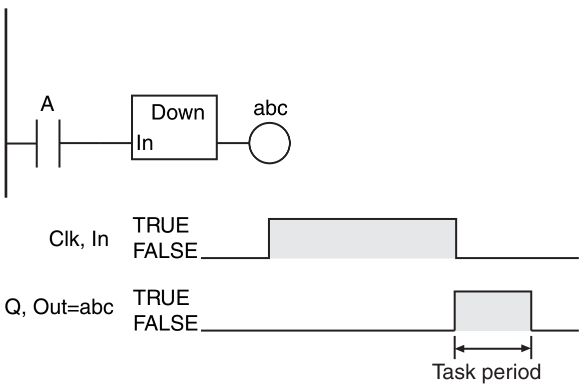
LD



ST

```
F_TRIG_instance(A, abc);
```

LD



Precautions for Correct Use

- Detection of upward or downward differentiation depends on differences between the current value of *Clk* or *In* and the value the last time the instruction was executed. Caution is required when using the JMP instruction or other times that the instruction is not executed every task period.
- If power is interrupted, the value of *Clk* or *In* is not detected as FALSE. The value of *Clk* or *In* is detected as FALSE only if the instruction evaluates the value of *Clk* or *In* while *Clk* or *In* is FALSE.
- In the first task period in which the Up instruction is executed, the value of *Out* is FALSE regardless of the value of *In*.
- If the value of *In* in the Up instruction is TRUE when the power supply is turned ON, the value of *Out* remains FALSE until the value of *In* changes to FALSE and then to TRUE.
- In the first task period in which the F_TRIG instruction is executed, the value of *Q* is FALSE regardless of the value of *Clk*.
- If the value of *Clk* in the F_TRIG instruction is FALSE when the power supply is turned ON, the value of *Q* remains FALSE until the value of *Clk* changes to TRUE and then to FALSE.
- In the first task period in which the Down instruction is executed, the value of *Out* is FALSE regardless of the value of *In*.
- If the value of *In* in the Down instruction is FALSE when the power supply is turned ON, the value of *Out* remains FALSE until the value of *In* changes to TRUE and then to FALSE.

Version Information

The value of *Q* when the R_TRIG instruction is executed and the value of *Clk* is TRUE depends on the unit version of the CPU Unit and the timing of execution of the instruction, as described in the following table.

Timing of execution of R_TRIG when *Clk* is TRUE	Value of *Q*	
	CPU Unit with unit version 1.02 or later	CPU Unit with unit version 1.01 or earlier
Task period in which R_TRIG is first executed	The value of *Q* is TRUE.	The value of *Q* is always FALSE.
When the power supply is turned ON	The value of *Q* is TRUE.	The value of *Q* remains FALSE until the value of *Clk* changes to FALSE and then to TRUE.

TestABit and TestABitN

TestABit: Outputs the value of the specified bit in a bit string.

TestABitN: Outputs the inverse of the value of the specified bit in a bit string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TestABit	Test A Bit	FUN	(@)TestABit EN — Out In Pos	Out:=TestABit (In, Pos);
TestABitN	Test A Bit NOT	FUN	(@)TestABitN EN — Out In Pos	Out:=TestABitN (In, Pos);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Bit string	Input	Bit string	Depends on data type.	---	*
Pos	Bit position		Specified bit position	0 to No. of bits in *In* – 1		0
Out	Bit value	Output	TestABit Value of specified bit TestABitN Inverse of value of specified bit	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Pos						OK														
Out	OK																			

Function

● TestABit

The TestABit instruction assigns the value of the bit at bit position *Pos* in the bit string *In* to the bit value *Out* when *EN* is TRUE.
When *EN* is FALSE, the value of *Out* is FALSE.

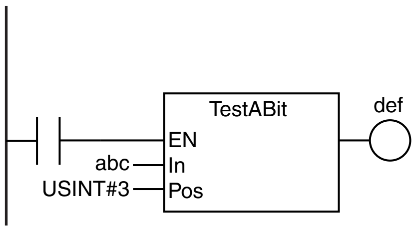
● TestABitN

The TestABitN instruction assigns the inverse of the value of the bit at bit position *Pos* in the bit string *In* to the bit value *Out* when *EN* is TRUE.

When *EN* is FALSE, the value of *Out* is FALSE.

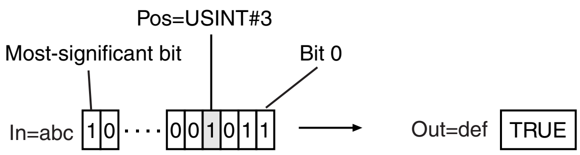
The following example shows the TestABit instruction when *Pos* is USINT#3.

LD



ST

```
def:=TestABit(abc, USINT#3);
```



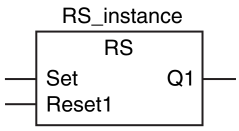
Precautions for Correct Use

- If this instruction is used in a ladder diagram, the value of *Out* changes to FALSE if an error occurs in the previous instruction on the rung.
- An error occurs in the following case. *Out* will be FALSE.
  - The value of *Pos* is greater than No. of bits in *In* – 1.

Sequence Output Instructions

RS

The RS instruction retains the value of a BOOL variable. It gives priority to the *Reset* input if both the *Set* input and *Reset* input are TRUE.

Instruction	Name	FB/FUN	Graphic expression	ST expression
RS	Reset-Priority Keep	FB		RS_instance(Set, Reset1, Q1);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Set*	Set	Input	Set input	Depends on data type.	---	0
Reset1*	Reset		Reset input			
Q1	Keep	Output	Keep output	Depends on data type.	---	---

* On Sysmac Studio version 1.03, you can use “S” instead of “Set” and “R1” instead of “Reset1” to more clearly show the correspondence between the variables and the parameter names in ST expressions. For example, you can use the following notation: *RS_instance*(*S:=A, R1:=B, Q1=>abc*);.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Set	OK																			
Reset1	OK																			
Q1	OK																			

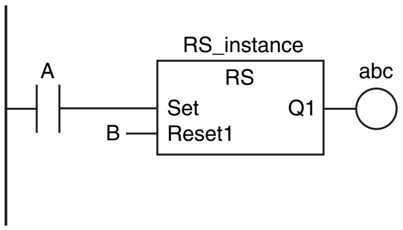
Function

The RS instruction forms a self-holding output that gives priority to resetting. The following table shows the relationship between the inputs are outputs.

Value of *Set*	Value of *Reset1*	Value of *Q1*
TRUE	TRUE	FALSE
TRUE	FALSE	TRUE
FALSE	TRUE	FALSE
FALSE	FALSE	Not changed.

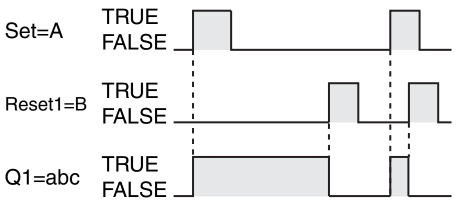
The following figure shows a programming example and timing chart.

LD



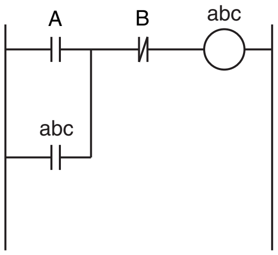
ST

```
RS_instance(A, B, abc);
```



Additional Information

- The RS instruction behaves like the following self-holding rung.



- However, if the RS instruction is in a master control region and the master control region is reset, the behavior will not be the same as the above self-holding rung.

Instruction/rung	Value of *B*	Value of *abc*
RS instruction	TRUE	Not changed.
	FALSE	FALSE
Self-holding rung	TRUE	FALSE
	FALSE	

Precautions for Correct Use

- Never use an NC bit directly from an external device for the *Reset1* input. The internal power supply in the Controller will not turn OFF immediately when the AC power is interrupted (even for momentary interruptions), and the input from the Input Unit may change to ON first. This could cause the *Reset1* input to change to TRUE.
- If this instruction is used in a ladder diagram, the value of *Q1* is retained if an error occurs in the previous instruction on the rung.
- If this instruction is not executed due to the execution of a jump instruction (e.g., the JMP instruction), *Q1* retains the value from the last execution.
- If this instruction is in a master control region and the master control region is reset, the operation is as follows:
  - If the value of *Reset1* is TRUE, the value of *Q1* is retained. If the value of *Reset1* is FALSE, the value of *Q1* changes to FALSE.
  - FALSE is input to the instruction that is connected to *Q1* even if the value of *Q1* is TRUE.

- Even if you connect a parameter with a Retain attribute to *Q1*, the value will not be retained when the power is interrupted. After the power supply is restored, the value of *Q1* will be FALSE when the operating mode is changed to RUN mode and the instruction is executed. If the self-holding rung given in *Additional Information* is used, the value is retained even after the power supply is restored.

SR

The SR instruction retains the value of a BOOL variable. It gives priority to the *Set* input if both the *Set* input and *Reset* input are TRUE.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SR	Set-Priority Keep	FB	SR_instance SR Set1 Q1 Reset	SR_instance(Set1, Reset, Q1);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Set1*	Set	Input	Set input	Depends on data type.	---	0
Reset	Reset		Reset input			
Q1	Keep	Output	Keep output	Depends on data type.	---	---

* On Sysmac Studio version 1.03, you can use "S1" instead of "Set1" and "R" instead of "Reset" to more clearly show the correspondence between the variables and the parameter names in ST expressions. For example, you can use the following notation: *SR_instance*(*S1:=A, R:=B, Q1=>ab*c);.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Set1	OK																			
Reset	OK																			
Q1	OK																			

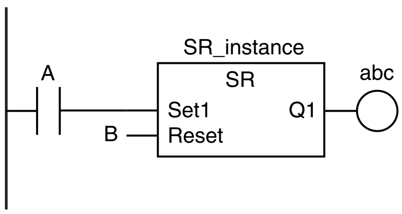
Function

The SR instruction forms a self-holding output that gives priority to setting. The following table shows the relationship between the inputs are outputs.

Value of *Set1*	Value of *Reset*	Value of *Q1*
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	FALSE
FALSE	FALSE	Not changed.

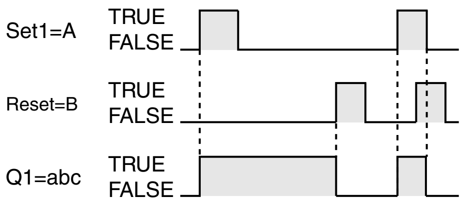
The following figure shows a programming example and timing chart.

LD



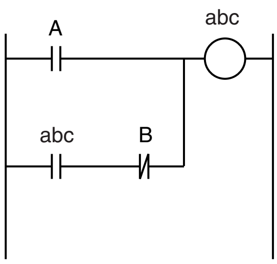
ST

```
SR_instance(A, B, abc);
```



Additional Information

- The SR instruction behaves like the following self-holding rung.



- However, if the SR instruction is in a master control region and the master control region is reset, the behavior will not be the same as the above self-holding rung.

Instruction/rung	Value of *B*	Value of *abc*
SR instruction	TRUE	Not changed.
	FALSE	FALSE
Self-holding rung	TRUE	FALSE
	FALSE	

Precautions for Correct Use

- Never use an NC bit directly from an external device for the *Reset* input. The internal power supply in the Controller will not turn OFF immediately when the AC power is interrupted (even for momentary interruptions), and the input from the Input Unit may change to ON first. This could cause the *Reset* input to change to TRUE.
- If this instruction is used in a ladder diagram, the value of *Q1* is retained if an error occurs in the previous instruction on the rung.
- If this instruction is not executed due to the execution of a jump instruction (e.g., the JMP instruction), *Q1* retains the value from the last execution.
- If this instruction is in a master control region and the master control region is reset, the operation is as follows:
  - If the value of *Reset* is TRUE, the value of *Q1* is retained. If the value of *Reset* is FALSE, the value of *Q1* changes to FALSE.
  - FALSE is input to the instruction that is connected to *Q1* even if the value of *Q1* is TRUE.

- Even if you connect a parameter with a Retain attribute to *Q1*, the value will not be retained when the power is interrupted. After the power supply is restored, the value of *Q1* will be FALSE when the operating mode is changed to RUN mode and the instruction is executed. If the self-holding rung given in *Additional Information* is used, the value is retained even after the power supply is restored.

Set and Reset

Set: Changes a BOOL variable to TRUE.
Reset: Changes a BOOL variable to FALSE.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Set	Set	---	Out —(S)— [Out —(PS)— Upward differentiation, Out —(NS)— Downward differentiation]	None
Reset	Reset	---	Out —(R)— [Out —(PR)— Upward differentiation, Out —(NR)— Downward differentiation]	None

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Out	Output	Output	Output	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out	OK																			

Function

● Set

The Set instruction changes *Out* to TRUE if the input is TRUE. If *Out* is TRUE, the Set instruction will not change it to FALSE even if the input changes to FALSE. Use the Reset instruction to change *Out* to FALSE.

● Reset

The Reset instruction changes *Out* to FALSE if the input is TRUE. If *Out* is FALSE, the Reset instruction will not change it to TRUE even if the input changes to FALSE. Use the Set instruction to change *Out* to TRUE.

The operation is as shown below if you do not specify upward or downward differentiation.

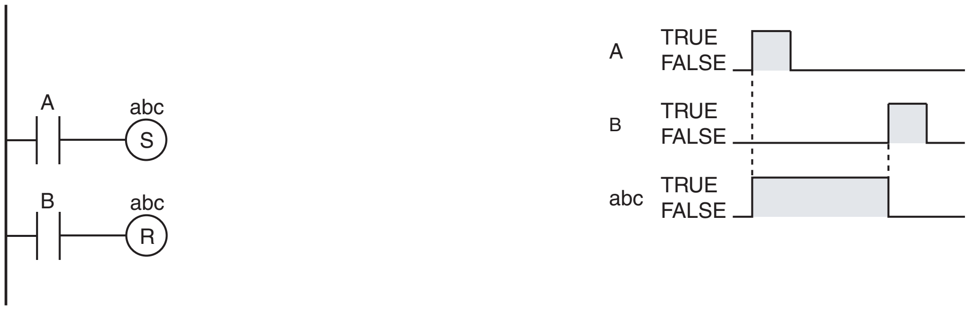
Instruction	Input	Output value
Set	TRUE	TRUE
	FALSE	Not changed.
Reset	TRUE	FALSE
	FALSE	Not changed.

If you specify upward or downward differentiation, the operation depends on the following: the value of the input for the last execution and the current value of the input. This is shown below.

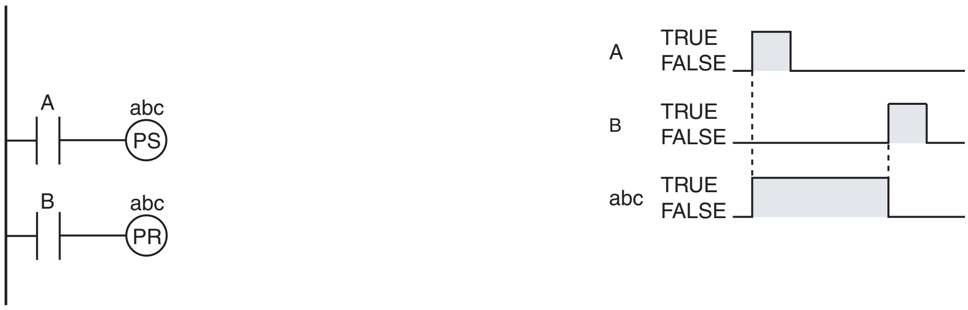
Instruction	Differentiation specification	Value of input at last execution and current value	Output value
Set	Upward differentiation	FALSE at the last execution → Currently TRUE	TRUE
		Other than the above.	Not changed.
	Downward differentiation	TRUE at the last execution → Currently FALSE	TRUE
		Other than the above.	Not changed.
Reset	Upward differentiation	FALSE at the last execution → Currently TRUE	FALSE
		Other than the above.	Not changed.
	Downward differentiation	TRUE at the last execution → Currently FALSE	FALSE
		Other than the above.	Not changed.

The following figure shows a programming example and timing chart.

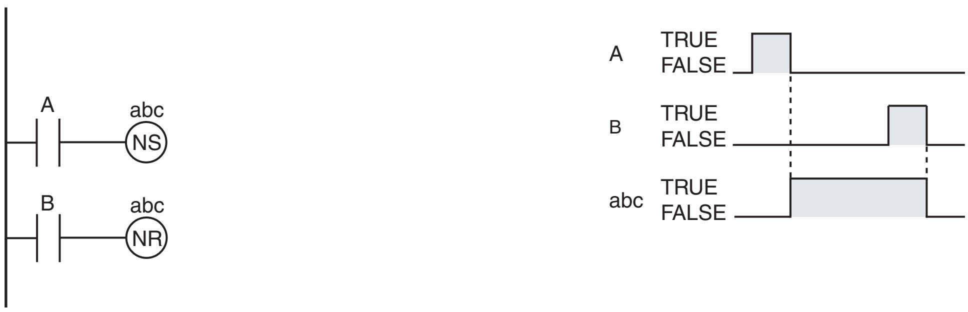
LD



LD



LD



Additional Information

Differences between the Set and Reset Instructions and the Out Instruction

- The Set and Reset instructions operate only when the input value changes to TRUE. They do not operate when the input value is FALSE. When the input value is FALSE, the output does not change.
- The Out instruction changes the specified variable to TRUE when the result from the previous instruction is TRUE and to FALSE when the result from the previous instruction is FALSE. It operates both when the input is TRUE and when it is FALSE.

Differences between the Set and Reset Instructions and the SR and RS Instructions

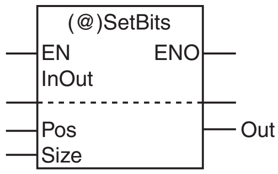
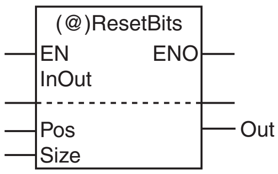
- The SR and RS instructions require that the *Set* input and *Reset* input are in the same place in the program. You can place the Set and Reset instructions in different places.

Precautions for Correct Use

- If this instruction is in a master control region and the master control region is reset, the value of *Out* is retained.
- If this instruction is not executed due to the execution of a jump instruction (e.g., the JMP instruction), the value of *Out* is retained.
- These instructions will not change the value of *Out* if you specify upward differentiation and the input is TRUE immediately after the power turns ON. The input must first change to FALSE and then to TRUE before the value of *Out* changes.
- These instructions will change the value of *Out* if you do not specify upward differentiation and the input is TRUE immediately after the power turns ON. In this case it is not necessary for the input to change to FALSE first.

SetBits and ResetBits

SetBits: Changes consecutive bits in bit string data to TRUE.
ResetBits: Changes consecutive bits in bit string data to FALSE.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SetBits	Set Bits	FUN		SetBits(InOut, Pos, Size);
ResetBits	Reset Bits	FUN		ResetBits(InOut, Pos, Size);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
InOut	Bit string	In-out	Bit string	Depends on data type.	---	---
Pos	Bit position	Input	Specified bit position	0 to No. of bits in *InOut* – 1	---	0
Size	Number of bits		Number of bits	0 to No. of bits in *InOut*		1
Out	Return value	Output	Always TRUE	TRUE only	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
InOut		OK	OK	OK	OK															
Pos						OK														
Size						OK														
Out	OK																			

Function

● SetBits

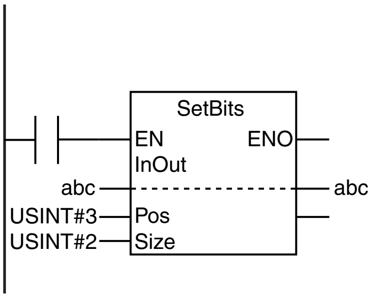
The SetBits instruction changes the value of *Size* bits from the bit position *Pos* in the bit string *InOut* to TRUE. The status of the other bits will not change.

- **ResetBits**

The ResetBits instruction changes the value of *Size* bits from the bit position *Pos* in the bit string *InOut* to FALSE. The status of the other bits will not change.

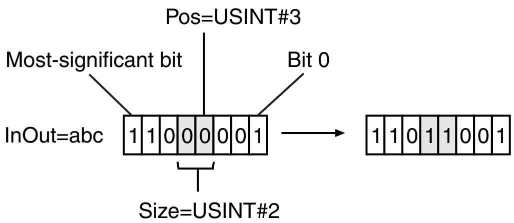
The following example shows the SetBits instruction when *Pos* is USINT#3 and *Size* is USINT#2.

LD



ST

```
SetBits(abc, USINT#3, USINT#2);
```



Additional Information

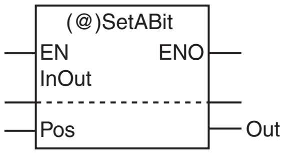
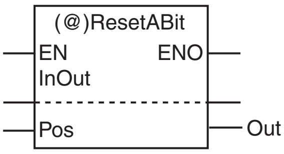
Use these instructions to globally set variables with AT specification in memory areas that handle data by word (e.g., the DM Area) to TRUE or FALSE.

Precautions for Correct Use

- If this instruction is in a master control region and the master control region is reset, the value of *InOut* is retained.
- If this instruction is not executed due to the execution of a jump instruction (e.g., the JMP instruction), the value of *InOut* is retained.
- The value of *InOut* does not change if the value of *Size* is 0.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* and *InOut* will not change.
  - The value of *Pos* is greater than No. of bits in *InOut* – 1.
  - The value of *Size* is outside of the valid range.
  - The value of *Pos* or *Size* exceeds the number of bits in *InOut*.

SetABit and ResetABit

SetABit: Changes the specified bit in bit string data to TRUE.
ResetABit: Changes the specified bit in bit string data to FALSE.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SetABit	Set A Bit	FUN		SetABit (InOut, Pos);
ResetABit	Reset A Bit	FUN		ResetABit (InOut, Pos);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
InOut	Bit string	In-out	Bit string	Depends on data type.	---	---
Pos	Bit position	Input	Specified bit position	0 to No. of bits in *InOut* – 1	---	0
Out	Return value	Output	Always TRUE	TRUE only	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
InOut		OK	OK	OK	OK															
Pos						OK														
Out	OK																			

Function

● SetABit

The SetABit instruction changes the value of the bit at bit position *Pos* in the bit string *InOut* to TRUE.

The bits that are not specified do not change.
Even if EN changes to FALSE after execution, the *Pos* bit in *InOut* will not change.

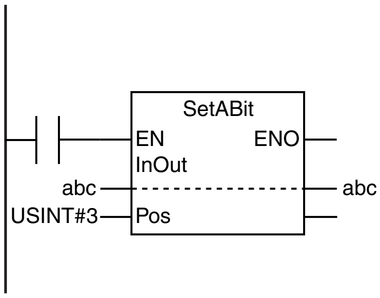
● ResetABit

The ResetABit instruction changes the value of the bit at bit position *Pos* in the bit string *InOut* to FALSE.

The bits that are not specified do not change.
Even if EN changes to FALSE after execution, the *Pos* bit in *InOut* will not change.

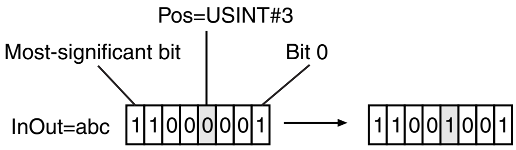
The following example shows the SetABit instruction when *Pos* is USINT#3.

LD



ST

```
SetABit(abc, USINT#3);
```



Additional Information

Differences between the SetABit and ResetABit Instructions and the OutABit Instruction

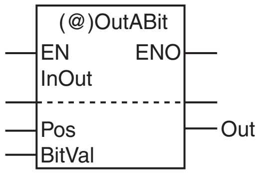
- The SetABit and ResetABit instructions change the value of the specified bit to either TRUE or FALSE.
- With the OutABit instruction, however, you can dynamically change the value to which the specified bit is set.

Precautions for Correct Use

- If this instruction is in a master control region and the master control region is reset, the value of *InOut* is retained.
- If this instruction is not executed due to the execution of a jump instruction (e.g., the JMP instruction), the value of *InOut* is retained.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following case. *ENO* will be FALSE, and *Out* and *InOut* will not change.
  - The value of *Pos* is greater than No. of bits in *In* – 1.

OutABit

The OutABit instruction changes the specified bit in bit string data to TRUE or FALSE.

Instruction	Name	FB/FUN	Graphic expression	ST expression
OutABit	Output A Bit	FUN		OutABit (InOut, Pos, BitVal);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
InOut	Bit string	In-out	Bit string	Depends on data type.	---	---
Pos	Bit position	Input	Specified bit position	0 to No. of bits in *InOut* – 1	---	0
BitVal	Set value		Value to set	Depends on data type.		TRUE
Out	Return value	Output	Always TRUE	TRUE only	---	---

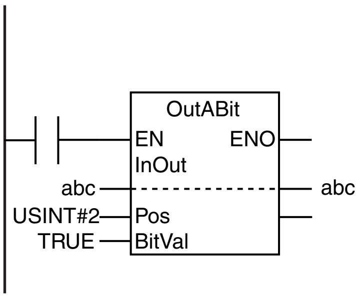
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
InOut		OK	OK	OK	OK															
Pos						OK														
BitVal	OK																			
Out	OK																			

Function

The OutABit instruction stores the value of set value *BitVal* at bit position *Pos* in the bit string *InOut*. Only the bit at *Pos* changes.

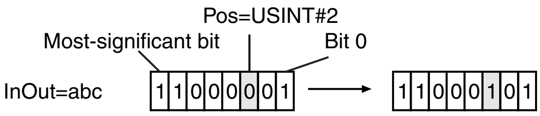
The following example is for when *Pos* is USINT#2 and *BitVal* is TRUE.

LD



ST

```
OutABit(abc, USINT#2, TRUE);
```



Additional Information

Differences between the SetABit and ResetABit Instructions and the OutABit Instruction

- The SetABit and ResetABit instructions change the value of the specified bit to either TRUE or FALSE.
- With the OutABit instruction, however, you can dynamically change the value to which the specified bit is set if you change the value of *BitVal*.

Precautions for Correct Use

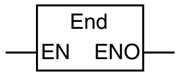
- If this instruction is in a master control region and the master control region is reset, the value of *InOut* is retained.
- If this instruction is not executed due to the execution of a jump instruction (e.g., the JMP instruction), the value of *InOut* is retained.
- Return value *Out* is not used when the instruction is used in ST.
- An error will occur in the following case. *ENO* will be FALSE, and *Out* and *InOut* will not change.
  - The value of *Pos* is greater than No. of bits in *InOut* – 1.

Sequence Control Instructions

Instruction	Name	Page
End	End	2-66
RETURN	Return	2-67
MC and MCR	Master Control Start/ Master Control End	2-68
JMP	Jump	2-80
FOR and NEXT	Repeat Start/ Repeat End	2-82
BREAK	Break Loop	2-89

End

The End instruction ends execution of a program in the current task period.

Instruction	Name	FB/FUN	Graphic expression	ST expression
End	End	FUN		None

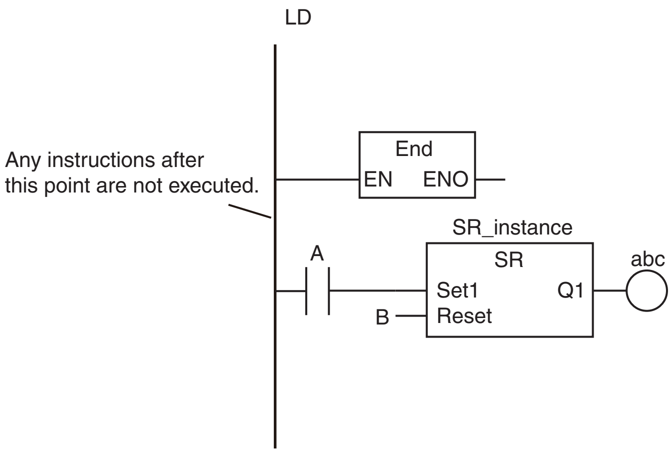
Variables

None

Function

The End instruction ends execution of a program in the current task period.

The following figure shows a programming example. When the End instruction is executed in the example, the SR instruction that follows it is not executed.



Precautions for Correct Use

- This instruction must be used only in a program.
- If this instruction is used in a function, function block, or inline ST, a building error will occur.
- You must connect this instruction to the left bus bar.

RETURN

The RETURN instruction ends a function or function block and returns processing to the calling instruction.

Instruction	Name	FB/FUN	Graphic expression	ST expression
RETURN	Return	FUN		RETURN;

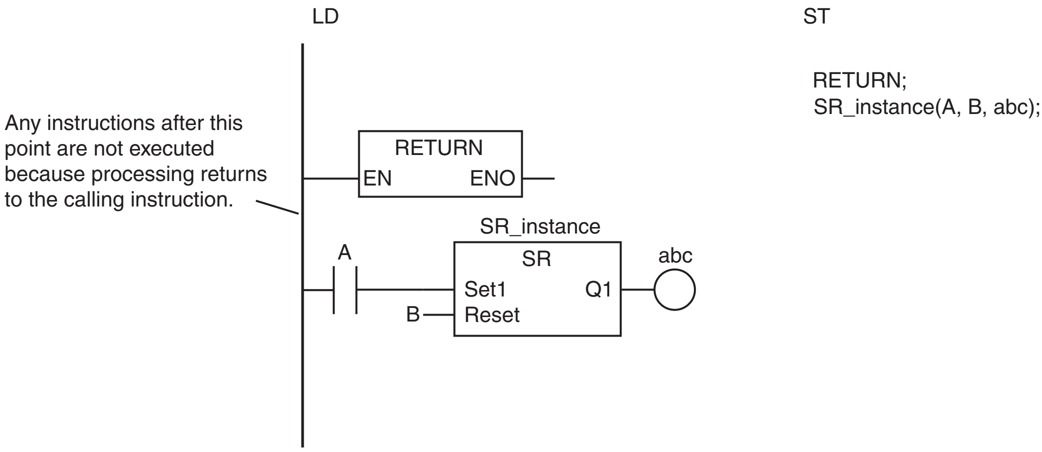
Variables

None

Function

The RETURN instruction ends a function or function block and returns processing to the calling instruction.

The following figure shows a programming example. When the RETURN instruction is executed in the example, the SR instruction that follows it is not executed.



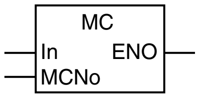
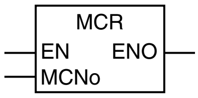
Precautions for Correct Use

- Observe the following precautions if you use this instruction in a ladder diagram.
  - Use this instruction only in functions and function blocks. If you use it in a program, a building error will occur.
  - Always connect this instruction directly to the left bus bar.
- Before you execute this instruction set the return values, output variables, and ENO value of the POU.
- If you use this instruction too often, the flow of processing will be difficult to understand. Use it with caution.

MC and MCR

MC: Marks the starting point of a master control region and resets the master control region.

MCR: Marks the end point of a master control region.

Instruction	Name	FB/FUN	Graphic expression	ST expression
MC	Master Control Start	---		None
MCR	Master Control End	---		None

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In (MC instruction only)	Master control input	Input	FALSE: Resets the master control region.	Depends on data type.	---	---
MCNo	Master control number		Master control number	0 to 14*		0

* The number is automatically registered by the Sysmac Studio. You do not need to set it.

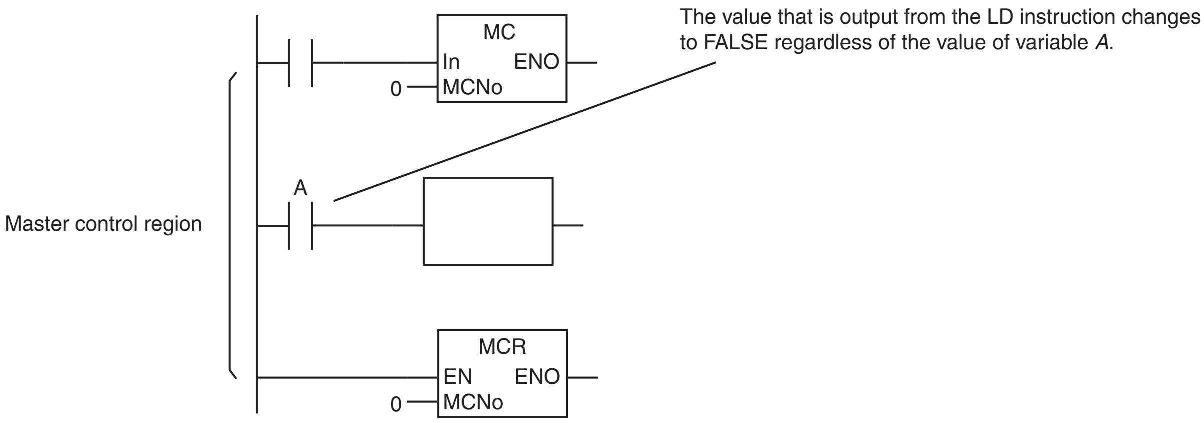
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In (MC instruction only)	OK																			
MCNo							OK													

Function

Master control is used to stop processing or place in an equivalent status all POUs in a specified region of a program. You can use master control to easily control the execution conditions for a relatively long segment of processing.

The region in the program for which master control is applied is called the master control region. You place the MC instruction at the start of the master control region and the MCR instruction at the end. When the value of the master control input *In* changes to FALSE, the outputs for all LD instructions that are connected to the left bus bar in the master control region are forced to change to FALSE. This is called a master control reset.

When master control is reset, the POUs that follow the LD instructions, as a rule, operate as if the execution condition is FALSE. There are, however, some POUs that operate differently. This is explained later.



If the value of *In* is TRUE, then master control is not reset. The POUs in the master control region operate normally.

POU Operation during a Master Control Reset

The operation of the POUs when master control is reset depends on the POU as described in the following table.

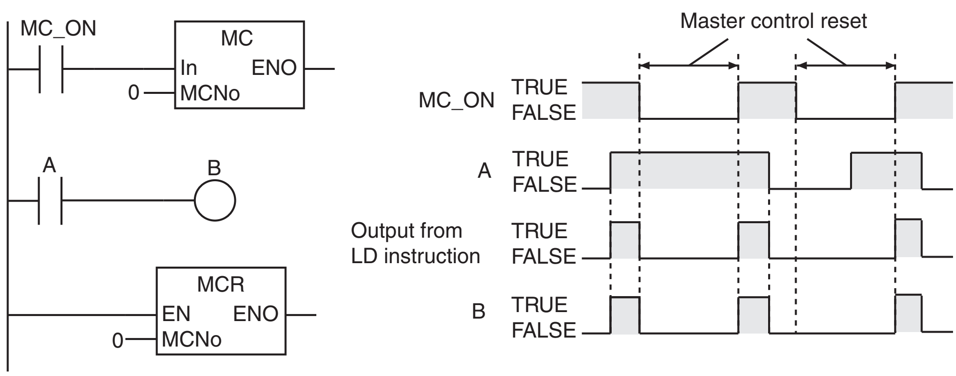
POU	Operation
Out and OutABit instructions	FALSE is output to the specified variable.
OutNot instruction	FALSE is output to the specified variable.
Set and Reset instructions	The output from before the master control reset is retained.
TON instruction	The instruction operates with a FALSE value for timer input *In*. That means that the timer is reset. The value of elapsed time *ET* changes to 0 and the value of timer output *Q* changes to FALSE.
TOF instruction	The instruction operates with a TRUE value for timer input *In*. That means that the timer is reset. The value of elapsed time *ET* changes to 0 and the value of timer output *Q* changes to TRUE. However, if an Out instruction is connected to *Q*, the execution condition to the Out instruction is FALSE.
TP instruction	The instruction operates with a FALSE value for timer input *In*. That means that the timer is reset. Timing active: The value of elapsed time *ET* is incremented to the end and then returns to 0. The value of timer output *Q* is TRUE until the end of timing, and then it changes to FALSE. Timing not active: The value of *ET* changes to 0 and the value of *Q* changes to FALSE. However, if an Out instruction is connected to *Q*, the execution condition to the Out instruction is FALSE even while timing is active.
AccumulationTimer instruction	The instruction operates with a FALSE value for timer input *In*. That means that the timer stops. The values of elapsed time *ET* and timer output *Q* are retained. However, if an Out instruction is connected to *Q*, the execution condition to the Out instruction is FALSE even if the value of *Q* is TRUE. However, reset *Reset* is enabled.
Timer instructions	The instruction operates with a FALSE value for timer input *In*. That means that the timer is reset. Remaining time *ET* is set to the value of set time *PT*, and the value of timer output *Q* changes to FALSE.
CTU, CTD, and CTUD instructions	These instructions are not executed. If the instruction was in operation before the master control reset, the count value from before the reset is held. If an Out instruction is connected to the Counter Completion Flag, *Q*, the execution condition to the Out instruction is FALSE.
JMP instruction	This instruction is not executed.

POU	Operation
FOR and NEXT instructions	These instructions are not executed.
BREAK instruction	This instruction is not executed.
Function blocks that are executed over more than one task period (i.e., instructions with *Done*, *Busy*, and *Error* output variables)	The power flow from the left bus bar changes to FALSE. If the instruction was operating before the master control reset, execution of the instruction is continued until processing is completed. *Busy, Done,* and *Error* outputs will be made, but FALSE will always be output if the next instruction is an output instruction. If a variable is directly connected to *Busy, Done*, or *Error*, the proper value for the instruction specifications will be assigned to that variable. You can also get the value of *Busy, Done*, or *Error* in the form *instance_name.output_variable.*
Other functions	These are not executed.
Other function blocks	The power flow from the left bus bar changes to FALSE.

The operation of some typical instructions is described below.

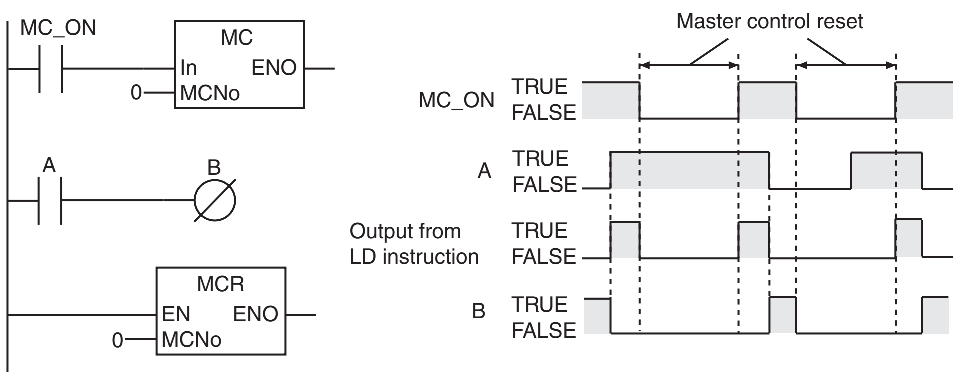
- **Out**

FALSE is output while the master control is reset.



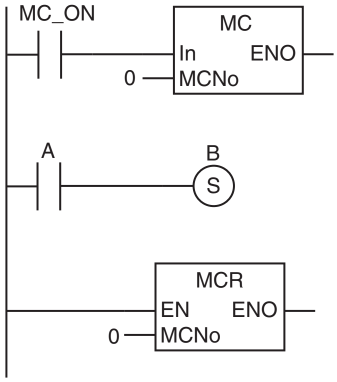
- **OutNot**

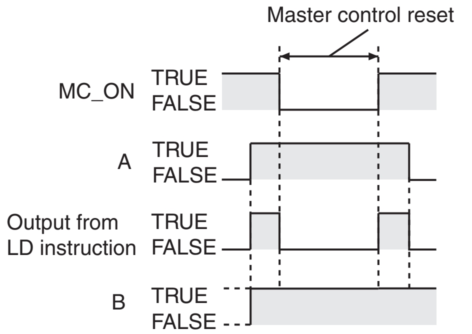
FALSE is output while the master control is reset. Caution is required because this operation of the OutNot instruction is different from when the output of the previous LD instruction is FALSE.

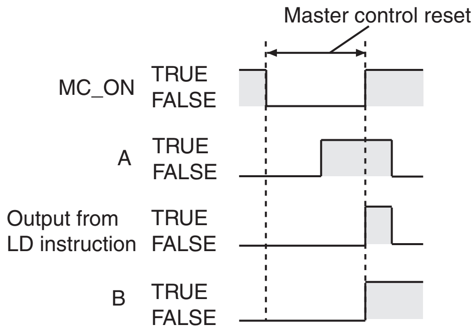


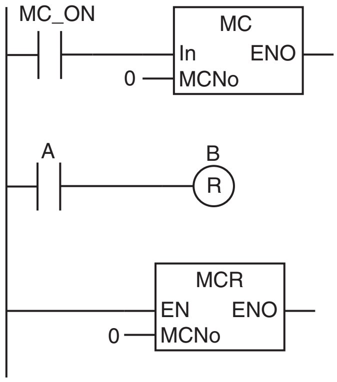
● Set and Reset

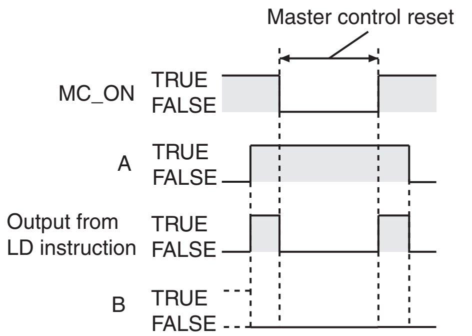
The previous value of the output is retained while the master control is reset.

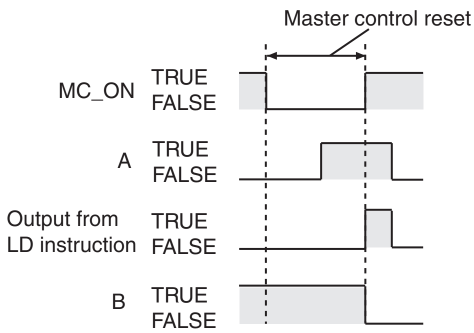






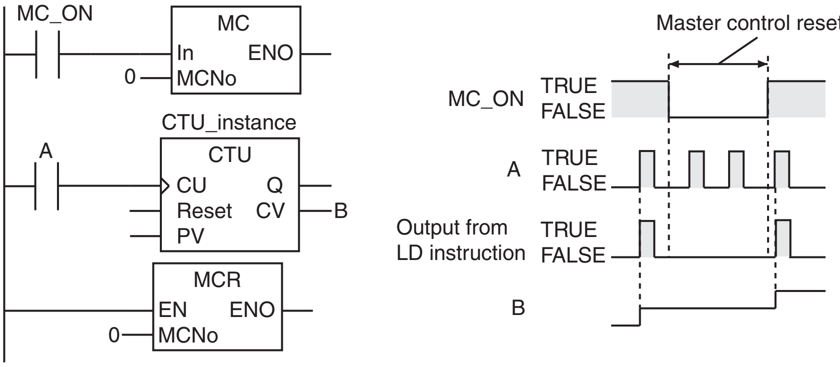






● CTU, CTD, and CTUD

The previous counter value is retained while the master control is reset. When the master control reset is cleared, counting continues from the counter value that was retained.



Operation of POUs with Input Upward Differentiation or Input Downward Differentiation

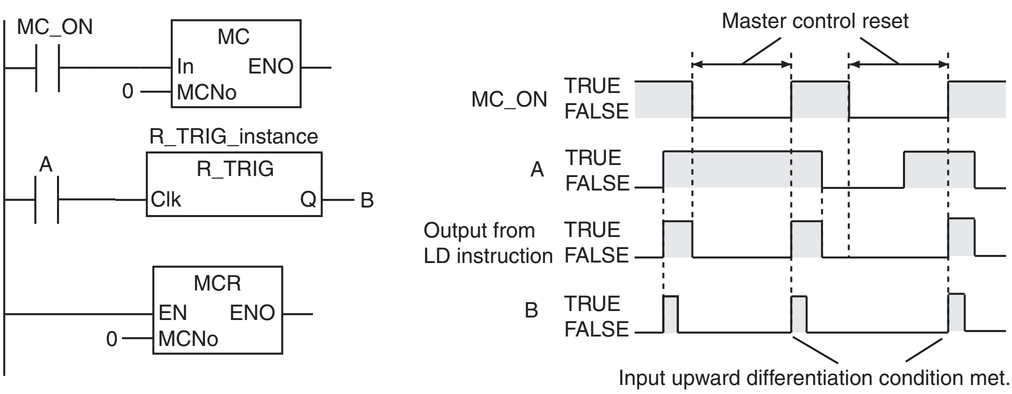
The POUs that are given in the following table have upward or downward differentiation specifications.

Differentiation	Instructions
Input upward differentiation	• LD, LDN, AND, ANDN, OR, ORN, and OUT with upward differentiation specifications • R_TRIG (Up) • Functions with an @ input upward differentiation option • Functions blocks (e.g., counter instructions) with input upward differentiation specifications
Input downward differentiation	• LD, LDN, AND, ANDN, OR, ORN, and OUT with downward differentiation specifications • F_TRIG (Down)

When the master control is reset or the reset is cleared, the execution conditions for these POUs change. That means that the upward or downward differentiation conditions for these POUs may be met. If the upward or downward differentiation conditions are met, then the instructions are executed accordingly. The operation of some typical instructions is described below.

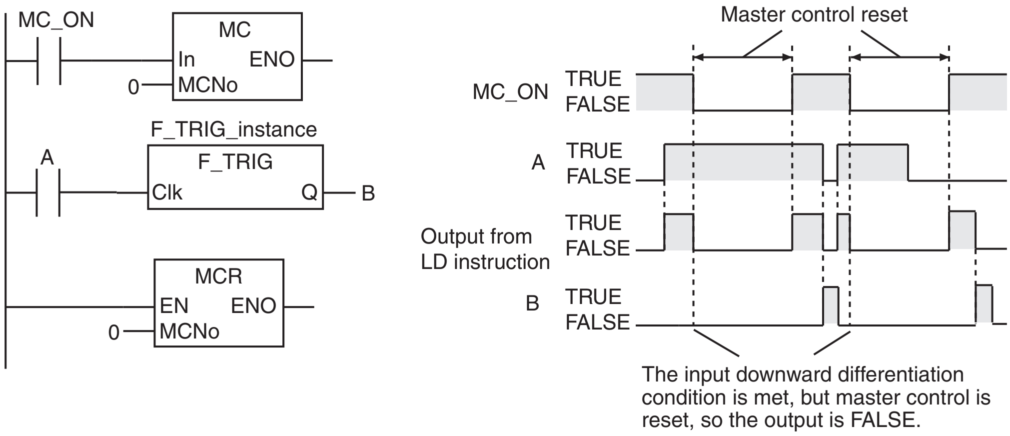
● R_TRIG (Up)

When the master control is reset, the execution condition changes to FALSE. If the execution condition is TRUE when the master control reset is cleared, the input upward differentiation condition is met and the instruction operates accordingly.



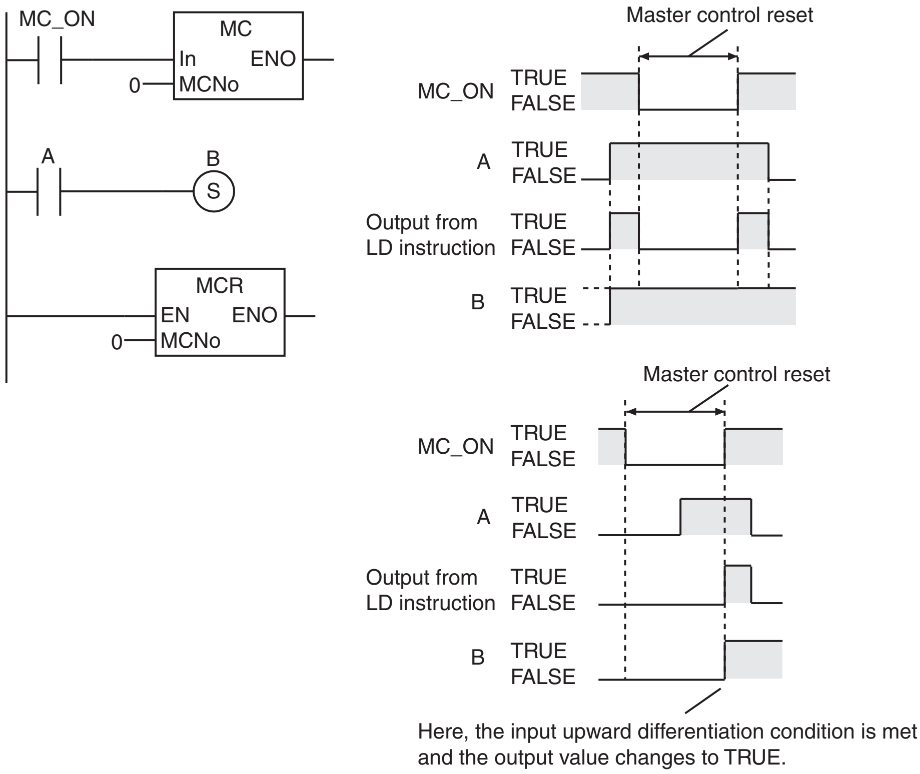
F_TRIG (Down)

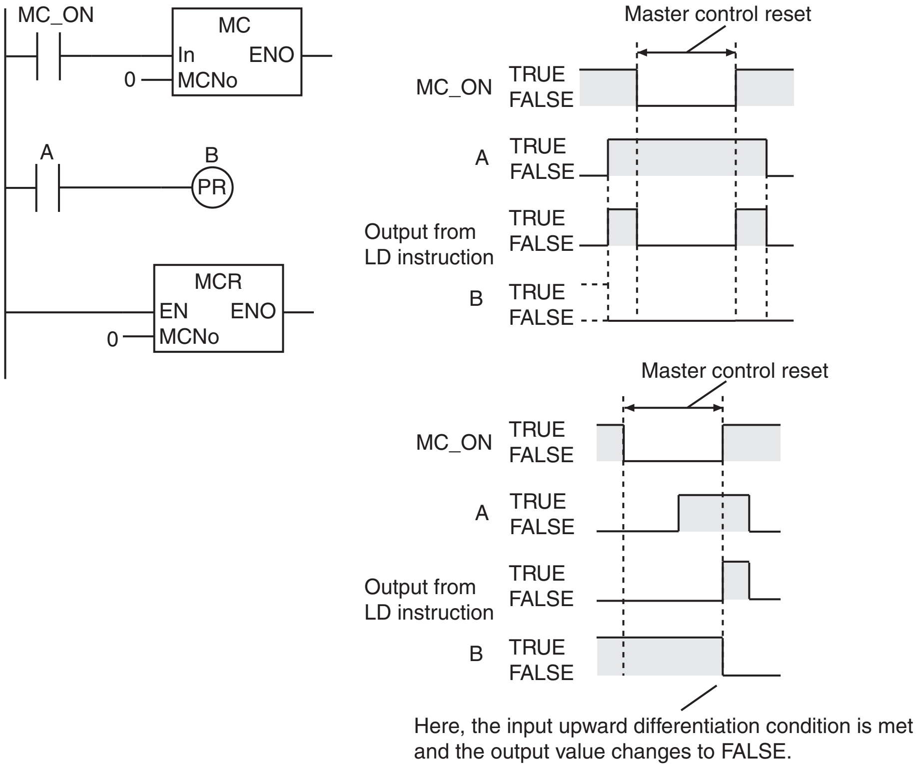
When the master control is reset, the execution condition changes to FALSE. If the previous execution condition was TRUE, then the input downward differentiation condition is met. However, the value of the output from the F_TRIG (Down) instruction during the master control reset is forced to change to FALSE, so the output value changes to FALSE.



Set and Reset with Input Upward Differentiation Specification

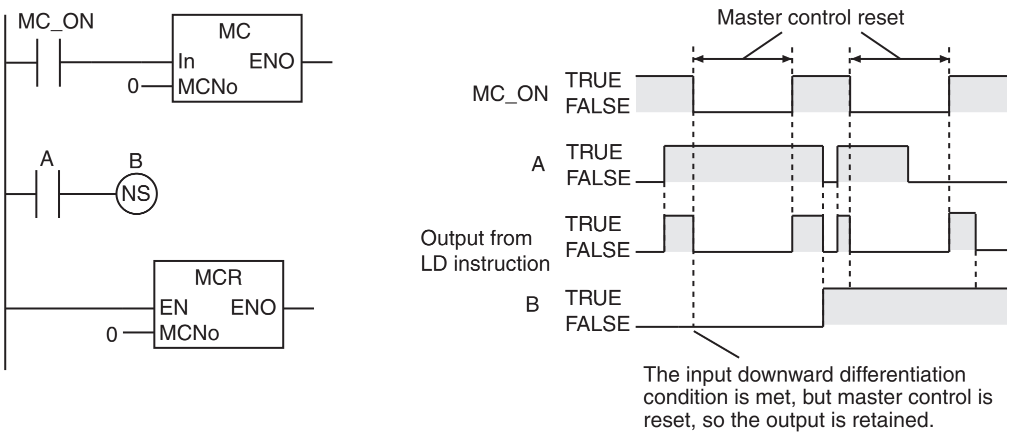
The previous value of the output is retained while the master control is reset. When the master control reset is cleared, the execution condition changes to TRUE and the instruction operates.

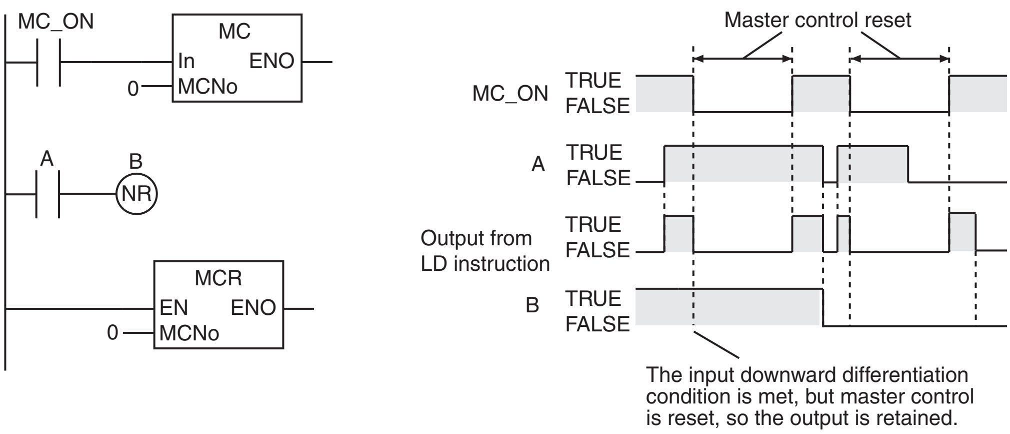




● Set and Reset with Input Downward Differentiation Specification

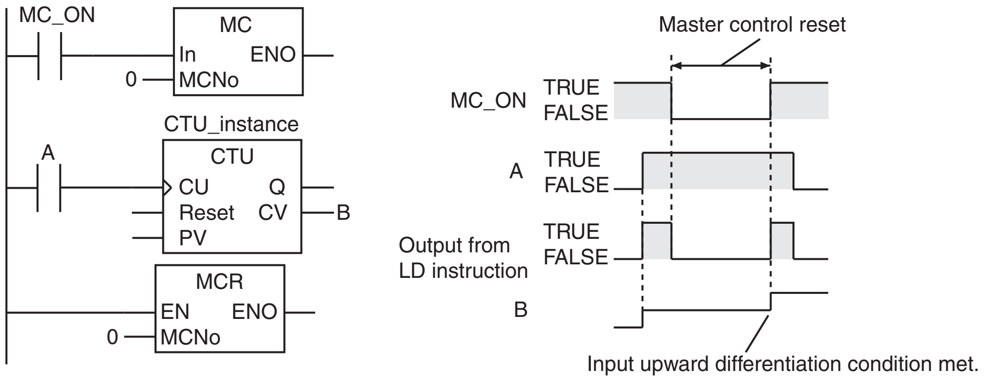
When the master control is reset, the execution condition changes to FALSE. If the previous execution condition was TRUE, then the input downward differentiation condition is met. However, during the master control reset, the previous output value is retained, so as a result the value of the output is retained.





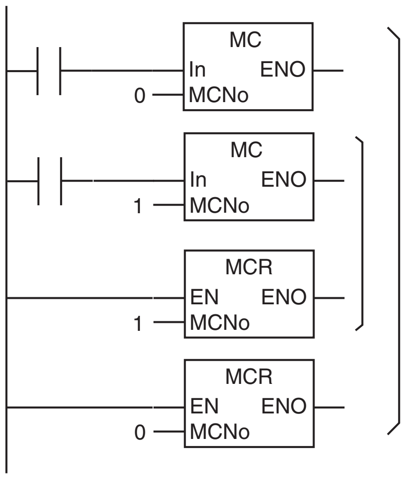
● CTU, CTD, and CTUD

When the master control is reset, the value of the counter input changes to FALSE. If the value of the counter input is TRUE when the master control reset is cleared, the input upward differentiation condition is met and the instruction counts.



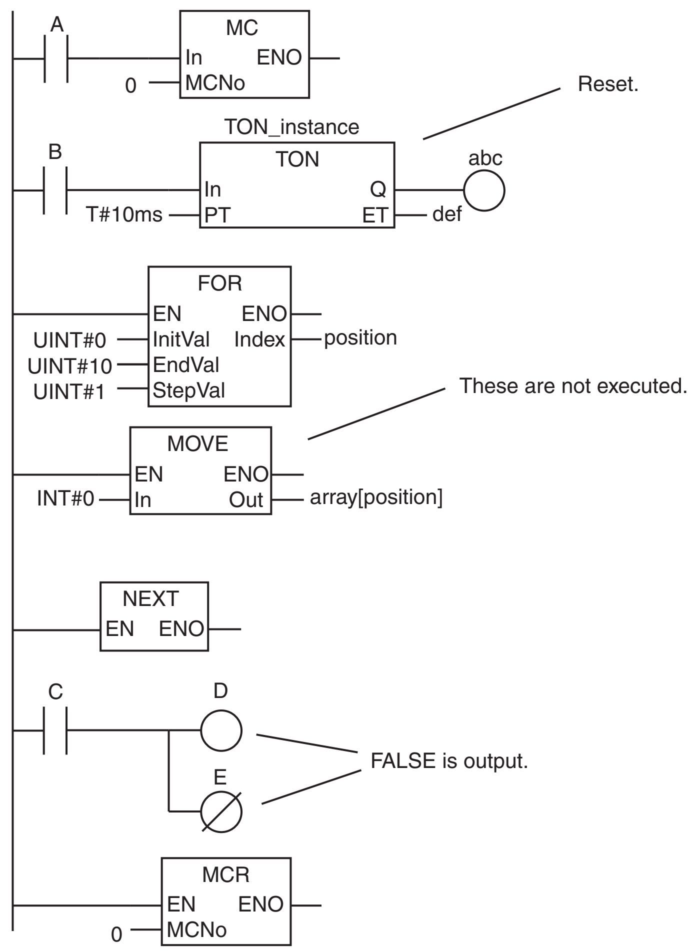
Always use the MC and MCR instructions as a pair in the same POU. The same value is used for master control number *MCNo* for both of the paired MC and MCR instructions. The user does not set the value of *MCNo*. It is automatically registered by the Sysmac Studio.

The MC and MCR instructions can be nested to up to 15 levels.



The following figure shows a programming example.

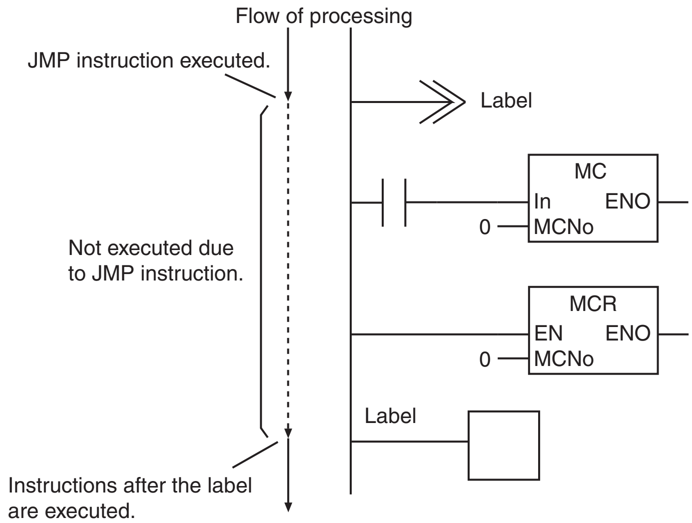
If the value of bit A is FALSE, the master control region is reset. While the master control region is in a reset state, the TON and MOVE instructions are not executed. Also the Out instruction and OutNot instruction will output FALSE to bits D and E.



Precautions for Correct Use

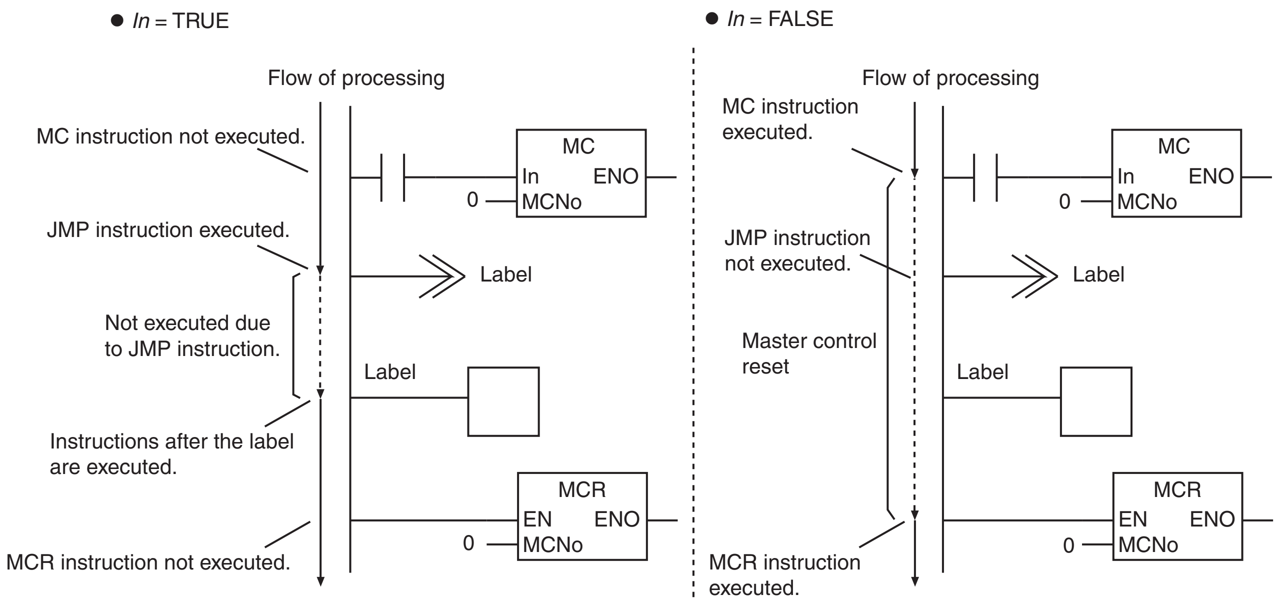
- These instructions must be used in a ladder diagram. They cannot be used in ST. They also cannot be used in inline ST in a ladder diagram.
- Always connect *In* directly to the left bus bar. You cannot pass a variable or constant to *In*.
- Always use the MC and MCR instructions as a pair in the same POU.
- Always place the MCR instruction after the MC instruction.
- Do not nest the MC and MCR instructions to more than 15 levels.
- If there is inline ST in the master control region, the inline ST is not executed when the master control region is reset.

- If you use the MC and MCR instructions and the JMP instruction together, the operation is as follows:
  - The following figure shows an MC-MCR pair inside a JMP-Label pair. Here, the jump is executed regardless of the value of *In*.

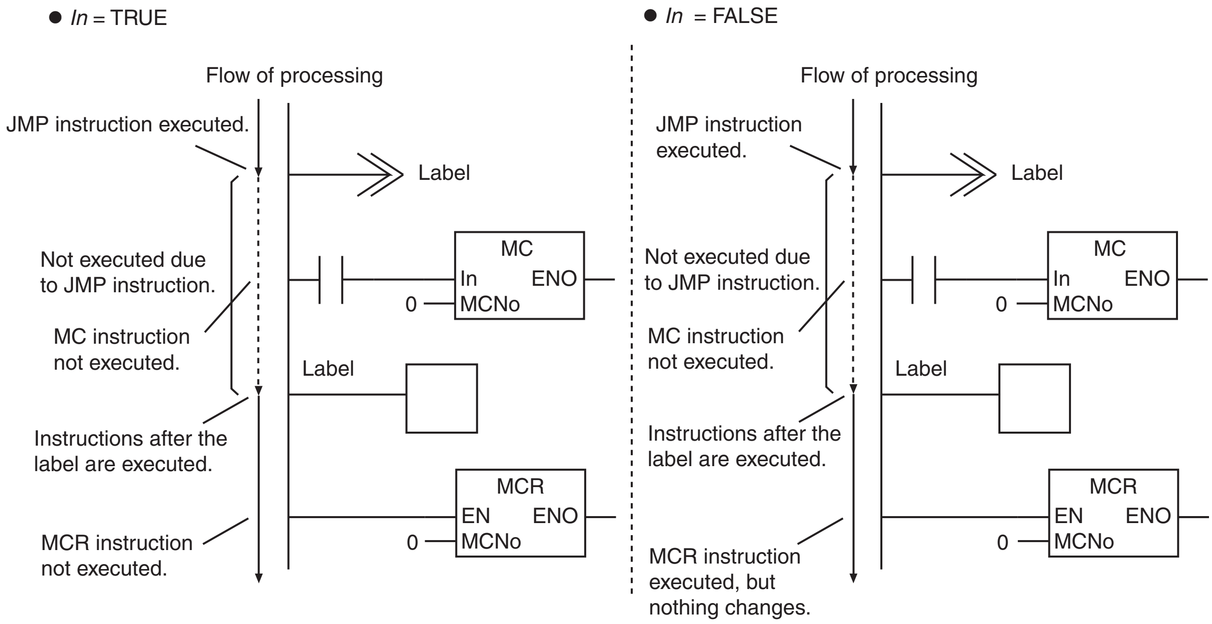


  - The following figure shows a JMP-Label pair inside an MC-MCR pair. Here, operation is as given in the following table.

Value of *In*	Operation
TRUE	Master control region is not reset. The jump is made.
FALSE	Master control region is reset. The jump is not made.

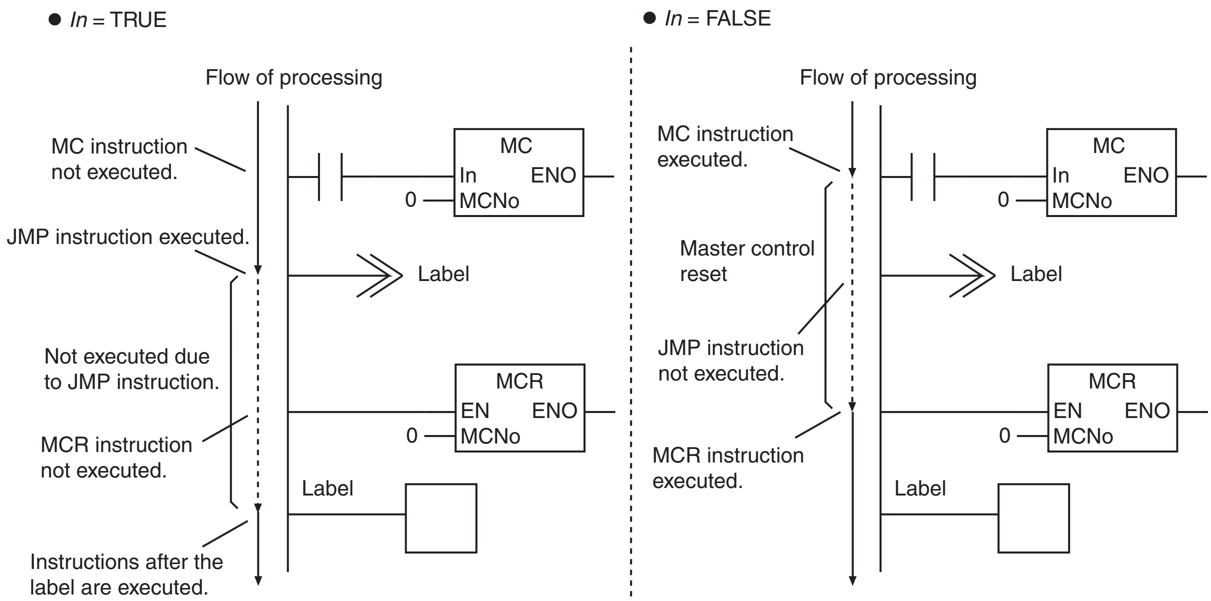


- The instructions are in the following order in the following figure: JMP instruction, MC instruction, Label, and MCR instruction. First, the jump is made. As a result, the MC instruction is not executed. Therefore, the instructions after the Label instruction are executed. If the value of *In* is FALSE, the MCR instruction is executed, but nothing changes.



- The instructions are in the following order in the following figure: MC instruction, JMP instruction, MCR instruction, and Label. Here, operation is as given in the following table.

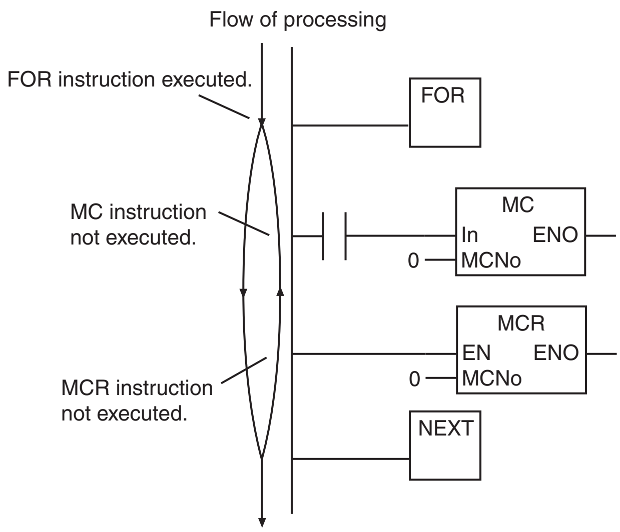
Value of *In*	Operation
TRUE	Master control region is not reset. The jump is made.
FALSE	Master control region is reset. The jump is not made.



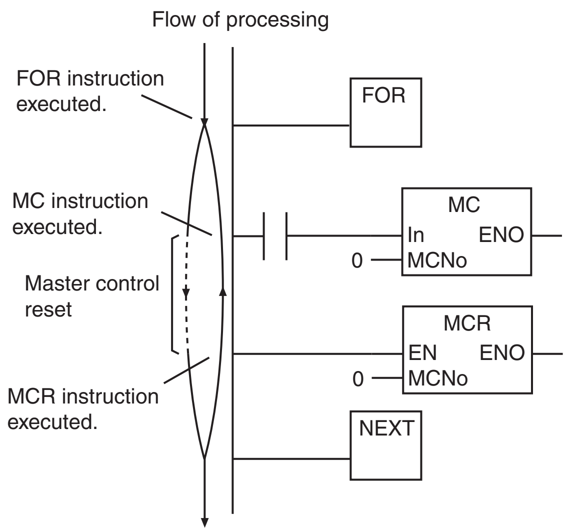
- If you use the MC and MCR instructions and the FOR and NEXT instructions together, the operation is as follows:
  - The following figure shows an MC-MCR pair inside a FOR-NEXT pair. Here, operation is as given in the following table.

Value of *In*	Operation
TRUE	Master control region is not reset. The FOR loop is executed.
FALSE	Master control region is reset. The FOR loop is executed, but the instructions between the MC and MCR instructions are not executed.

● *In* = TRUE



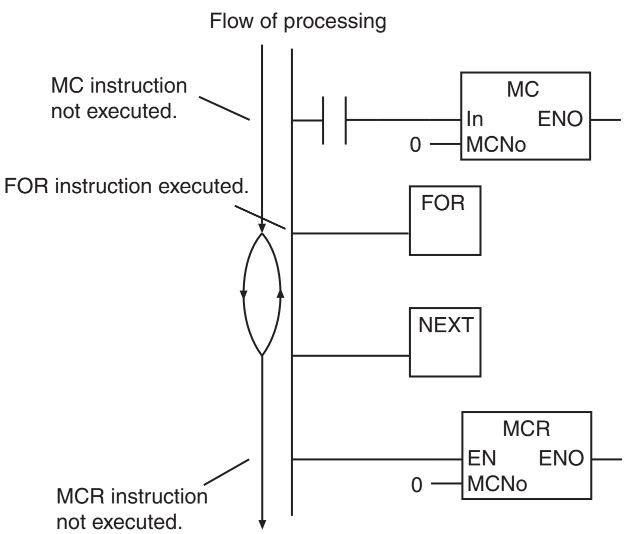
● *In* = FALSE



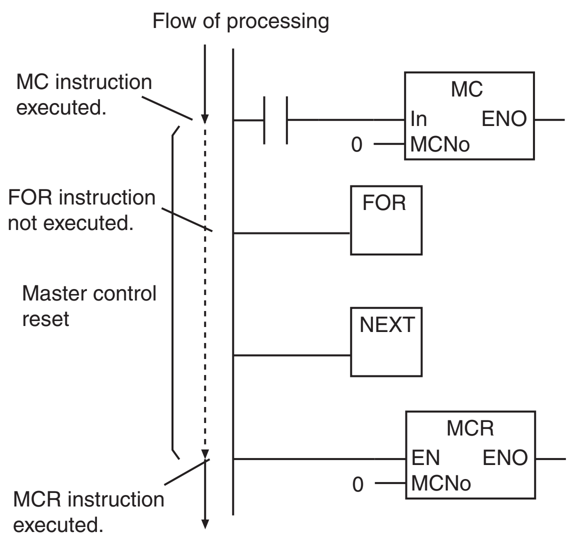
  - The following figure shows a FOR-NEXT pair inside an MC-MCR pair. Here, operation is as given in the following table.

Value of *In*	Operation
TRUE	Master control region is not reset. The FOR loop is executed.
FALSE	Master control region is reset. The FOR loop is not executed.

● *In* = TRUE



● *In* = FALSE



- A building error occurs if the FOR, NEXT, MC, and MCR instructions are used in either of the following orders.

  FOR, MC, NEXT, MCR, or MC, FOR, MCR, NEXT

JMP

The JMP instruction moves processing to the specified jump destination.

Instruction	Name	FB/FUN	Graphic expression	ST expression
JMP	Jump	FUN		None

Variables

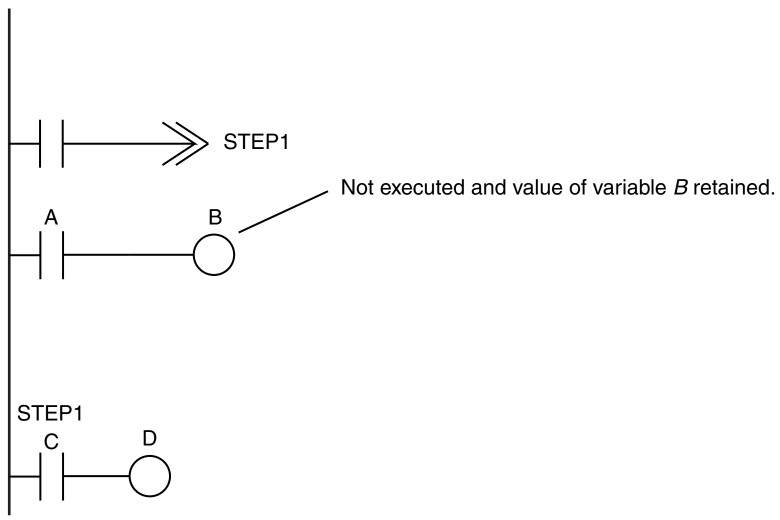
None

Function

When the execution condition is TRUE, the JMP instruction moves processing to the jump destination specified by a Label in a ladder diagram. The label can be any text string.

The following figure shows a programming example. This example uses the text string *STEP1* as the label. When the JMP instruction is executed, processing moves to the location marked *STEP1*. In this example, the Out instruction between the JMP instruction and the Label is not executed, and the value of variable *B* is retained.

LD



Additional Information

- You can also jump to a Label instruction above the JMP instruction in the section.
- You can use the same Label instruction as the jump destination for more than one JMP instruction.

Precautions for Correct Use

- You cannot omit labels. If you omit a label, a building error will occur.
- Place the JMP and Label instructions in the same POU and in the same section.
- Do not set the same Label instruction more than once in the same section.
- You cannot jump into a FOR-NEXT loop from outside the loop.

- The following restrictions apply to the characters that can be used as labels.

Item	Specification
Maximum number of bytes	127 bytes 127 characters when converted to ACSII 31 characters when converted to Japanese characters (including single-byte kana)
Character code	UTF-8
Applicable characters	Not case sensitive. English alphanumeric characters and other language characters. Symbols: _ (underbar) and ~ (tilda)
Prohibited text strings	• Any text string that starts with ASCII characters 0 to 9 (character codes 16#30 to 16#39) • A text string that consists of only a single _ (underbar) ASCII character • Any text string that includes two or more consecutive _ (underbar) ASCII characters • Any text string that starts with an _ (underbar) ASCII character • Any text string that ends with an _ (underbar) ASCII character • Any text string that starts with ‘P_’
Prohibited characters	Blank space ! " # $ & ' ( ) * + , - . / : ; < = > ? @ [ ] ^ ` %

- Variable names cannot be used as labels.

FOR and NEXT

FOR: Marks the starting position for repeat processing and specifies the repeat condition.

NEXT: Marks the ending position for repeat processing.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FOR	Repeat Start	FUN	FOR EN ENO InitVal Index EndVal StepVal	FOR Index:=InitVal TO EndVal BY StepVal DO expression END_FOR*; * In ST, do not use NEXT to mark the ending position of repeat processing. Use END_FOR instead.
NEXT	Repeat End	FUN	NEXT EN ENO	

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
InitVal	Initial value	Input	Value to set the *Index* to when repetition is started.	Depends on data type.*1	---	*2
EndVal	End value		Value of Index where repetition is stopped			
StepVal	Increment		Value to add to *Index* each time processing is repeated	Depends on the data type.*3		*4
Index	Control variable	Output	Loop index	Depends on data type.	---	---

*1 When using a ladder diagram, set *InitVal* so that it is less than *EndVal*.

*2 If you omit an input parameter, the default value is not applied. A building error will occur.

*3 When using a ladder diagram, 0 and negative numbers are not included. When using an ST program, 0 is not included.

*4 If you omit the input parameter in a ladder diagram, the default value is not applied. A building error will occur. If you omit the input parameter in ST, a default value of 1 is applied.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
InitVal						OK	OK	OK	OK	OK	OK	OK	OK							
	An enumeration, array element or structure member can also be specified.*																			
EndVal						OK	OK	OK	OK	OK	OK	OK	OK							
	An array element or structure member can also be specified.																			
StepVal						OK	OK	OK	OK	OK	OK	OK	OK							
	An array element or structure member can also be specified.																			
Index						OK	OK	OK	OK	OK	OK	OK	OK							
	An array element or structure member can also be specified.																			

* You cannot specify enumerations in ladder diagrams.

Function

The FOR and NEXT instructions repeat the processing that you place between them. (FOR and END_-FOR are used in ST.) The processing procedure for a FOR-NEXT loop is as follows:

1. The value of *InitVal* is set in control variable *Index*.
2. The values of *StepVal*, *Index*, and *EndVal* are checked to see if the conditions in the following table are met. If the conditions are met, processing moves to step 3. If the conditions are not met, repeat processing is not performed, and processing moves to the next process after the NEXT instruction (or END_FOR in ST).

Programming language	Conditions to start repeat processing
Ladder diagram	StepVal ≥ 0 and Index < EndVal
ST	StepVal ≥ 0 and Index ≤ EndVal
	StepVal < 0 and Index ≥ EndVal

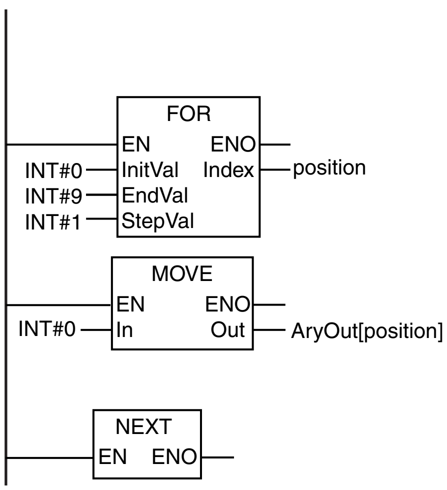
3. The values of *Index* and *EndVal* are checked to see if the conditions in the following table are met. If the conditions are met, processing moves to step 4. If the conditions are not met, repeat processing is ended, and processing moves to the next process after the NEXT instruction (or END_FOR in ST).

Programming language	Conditions to continue repeat processing
Ladder diagram	*Index* ≤ *EndVal*
ST	If *StepVal* ≥ 0, *Index* must be ≤ *EndVal*
	If *StepVal* < 0, *Index* must be ≥ *EndVal*

4. The processing between the FOR instruction and the NEXT instruction (or the END_FOR instruction in ST) is executed once.
5. The value of *StepVal* is added to *Index*.
6. Processing returns to step 3.

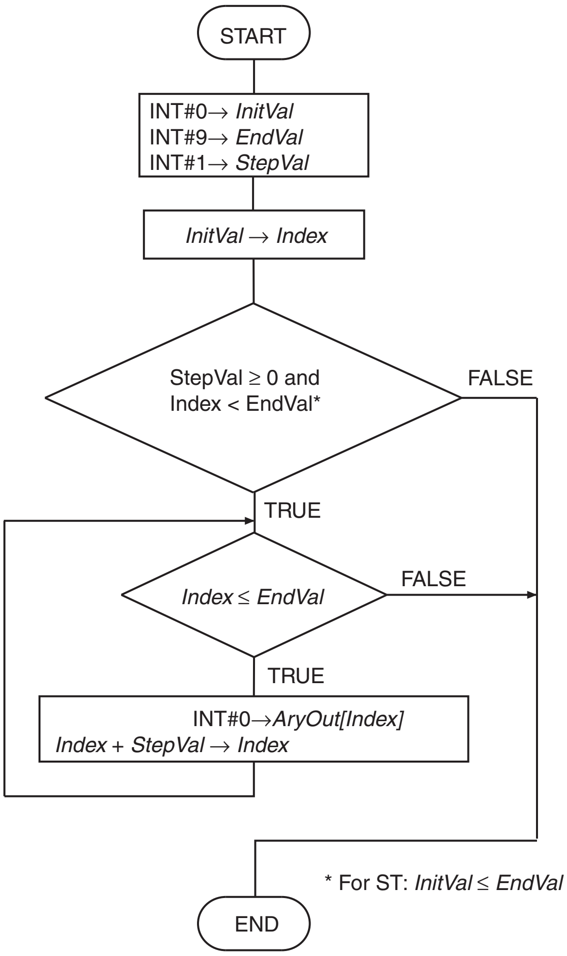
The following example is for when *InitVal* is INT#0, *EndVal* is INT#9, and *StepVal* is INT#1. The MOVE instruction is executed 10 times and INT#0 is assigned to array variables *AryOut[0] to AryOut[9]*.

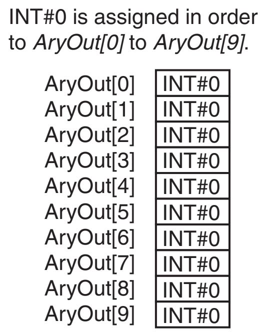
LD



ST

```
FOR position:=INT#0 TO INT#9 BY INT#1 DO
        AryOut[position]:=INT#0;
END_FOR;
```





ST Programming Example That Uses Expressions or Functions for Input Variables.

If you use these instructions in an ST program, you can use the following notation for the *InitVal, EndVal,* and *StepVal* input variables.

- An expression with an integer result
- A function that returns an integer
- A function that returns an enumerator

The following example programs a function for *EndVal* and an expression for *StepVal.*

```
A := DINT#1;
B := DINT#2;
C := REAL#9.6;
FOR i := 0 TO RoundUp(C) BY A+B DO
   DINTArray[i] := i;
END_FOR;
```

Version Information

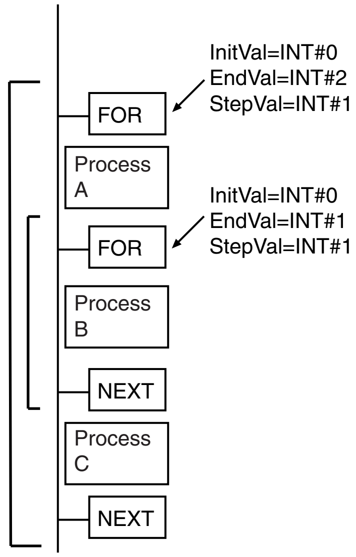
Sysmac Studio version 1.08 or higher is required to use expressions for *EndVal* and *StepVal.*

You can use an expression for *InitVal* even with Sysmac Studio version 1.07 or lower.

Additional Information

- Execute a BREAK instruction (or an EXIT instruction in ST) to cancel repeat processing. The processing between the BREAK instruction and the NEXT instruction will not be executed.
- FOR-NEXT loops (or FOR-END_FOR loops in ST) can be nested. In the following figure, the processes are performed in the following order.

Process A → Process B → Process B → Process C → Process A → Process B → Process B → Process C → Process A → Process B → Process B → Process C



Precautions for Correct Use

- In a ladder diagram, connect the FOR and NEXT instructions directly to the left bus bar.
- Always use the FOR and NEXT instructions (FOR and END_FOR statements in ST) as a pair. A programming error will occur if there is not the same number of both instructions.
- Program the paired FOR and NEXT instructions in the same section.
- Set the condition to end repetition carefully so that you do not create an infinite loop. If an infinite loop occurs, task execution will time out.

Example: If the values that are given in the following table are used for the input parameters to the variables, the value of Index will never be greater than the value of *EndVal* because the maximum value of SINT data is 127. Therefore, an infinite loop is created. Do not set the maximum value for the data type in *EndVal*.

Variable	Value of input parameter
InitVal	SINT#0
EndVal	SINT#127
StepVal	SINT#1
Index	---

- The following table describes operation according to the values of *StepVal*, *InitVal*, and *EndVal*.

Programming language	Value of *StepVal*	Values of *InitVal* and *EndVal*	Operation
Ladder diagram	*StepVal* > 0	*InitVal* < *EndVal*	Operation is normal.
		InitVal ≥ *EndVal*	The processing between the FOR and NEXT instructions is not executed even once. An error does not occur.
	StepVal < 0	*InitVal* < *EndVal*	The processing between the FOR and NEXT instructions is executed an indeterminate number of times. Do not use settings like these. An error does not occur.
		InitVal ≥ *EndVal*	The processing between the FOR and NEXT instructions is not executed even once. An error does not occur.
	StepVal = 0	*InitVal* < *EndVal*	An infinite loop occurs and task execution times out.
		InitVal ≥ *EndVal*	The processing between the FOR and NEXT instructions is not executed even once. An error does not occur.
ST	*StepVal* > 0	*InitVal* ≤ *EndVal*	Operation is normal.
		InitVal > *EndVal*	The processing between the FOR and END_FOR instructions is not executed even once. An error does not occur.
	StepVal < 0	*InitVal* < *EndVal*	The processing between the FOR and END_FOR instructions is not executed even once. An error does not occur.
		InitVal ≥ *EndVal*	Operation is normal.
	StepVal = 0	*InitVal* ≤ *EndVal*	An infinite loop occurs and task execution times out.
		InitVal > *EndVal*	The processing between the FOR and END_FOR instructions is not executed even once. An error does not occur.

- The FOR-NEXT loops can be nested up to 15 levels, but count all nesting levels for the following instructions: IF, CASE, FOR, WHILE, and REPEAT.
- If loops are nested, you will need one BREAK instruction (or one EXIT instruction in ST) for each nesting level to cancel all repeat processing.
- Do not use Jump Instructions (e.g., the JMP instruction) to interrupt repeat processing. Always use a BREAK instruction (or an EXIT instruction in ST) to cancel repeat processing.
- The operation to change the values of *InitVal, EndVal,* and *StepVal* during repeat processing is different in a ladder diagram and ST.

Variable	Operation	
	Ladder diagram	ST
InitVal	The new value is not applied until repeat processing is completed.	The new value is not applied until repeat processing is completed.
EndVal	The new value is applied even during repeat processing.	
StepVal	The intended operation may not occur. Do not change the value of this variable during repeat processing.	

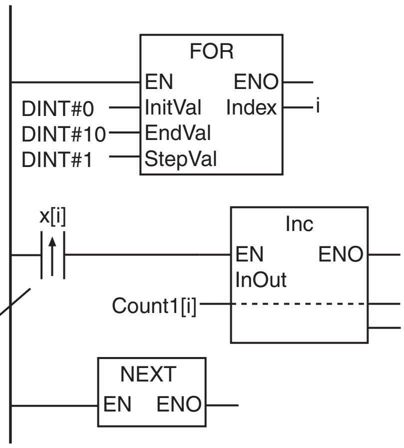
- In a ladder diagram, use the same data type for *InitVal*, *EndVal*, *StepVal*, and *Index*. Otherwise, a building error will occur.
- Set the data type of *Index* to include the valid ranges of *InitVal, EndVal,* and *StepVal*. Otherwise, a building error will occur.

- The value of Index after repeat processing is different in a ladder diagram and ST. In a ladder diagram, the value of *StepVal* is not added to *Index* at the end of repeat processing. In ST, the value of *StepVal* is added to *Index* at the end of repeat processing. Processing is repeated the same number of times.

  The following example is for when *InitVal* is 1, *EndVal* is 100 and *StepVal* is 1.
  Ladder diagram: The value of *Index* will be 100 after 100 repetitions.
  ST: The value of *Index* will be 101 after 100 repetitions.
- Caution is required when you specify upward or downward differentiation for a LD, AND, or OR instruction in a FOR loop in a ladder diagram and an array is used for the LD, AND, or OR instruction.

  For upward or downward differentiation, the value of the specified variable at the previous execution is compared with the value of the specified variable at the current execution to determine upward or downward differentiation. Normally, the value of the specified variable does not change every time the instruction is executed. However, if an array is specified in a FOR loop, the array element changes each time the instruction is executed. Therefore, upward or downward differentiation is determined by comparing different array elements.
  The following table shows the relationship between the values of *x[i-1]* and *x[i]*, and the increment processing for *Count1[i]*.

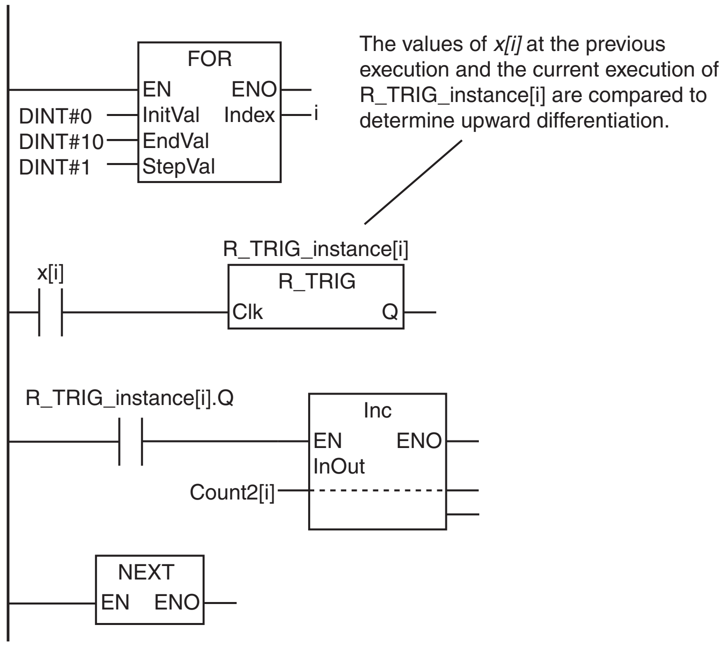
Value of *x[i-1]*	Value of *x[i]*	Increment processing for *Count1[i]*
TRUE	TRUE	Not executed.
TRUE	FALSE	Not executed.
FALSE	TRUE	Executed.
FALSE	FALSE	Not executed.



Upward differentiation is detected by comparing *x[i]* to *x[i–1]*.

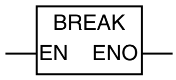
- In the following programming, upward differentiation of *x[i]* is detected by the R_TRIG instruction. An instance of the R_TRIG instruction is provided for each element of *x[i],* so it is possible to detect the elements of *x[i]* for which there was upward differentiation. The following table shows the relationship between the value of *x[i]* for the previous execution of R_TRIG_instance[i], the value of *x[i]* for the current execution of R_TRIG_instance[i], and the increment processing of *Count2[i]*.

Value of *x[i]* for previous execution of R_TRIG_instance[i]	Value of *x[i]* for current execution of R_TRIG_instance[i]	Increment processing for *Count2[i]*
TRUE	TRUE	Not executed.
TRUE	FALSE	Not executed.
FALSE	TRUE	Executed.
FALSE	FALSE	Not executed.


FOR
EN
ENO
DINT#0
InitVal
Index
i
DINT#10
EndVal
DINT#1
StepVal
The values of *x[i]* at the previous execution and the current execution of R_TRIG_instance[i] are compared to determine upward differentiation.
R_TRIG_instance[i]
x[i]
R_TRIG
Clk
Q
R_TRIG_instance[i].Q
Inc
EN
ENO
InOut
Count2[i]
NEXT
EN
ENO

BREAK

The BREAK instruction is used to cancel repeat processing from the lowest level FOR instruction to the NEXT instruction.

Instruction	Name	FB/FUN	Graphic expression	ST expression
BREAK	Break Loop	FUN		None

Variables

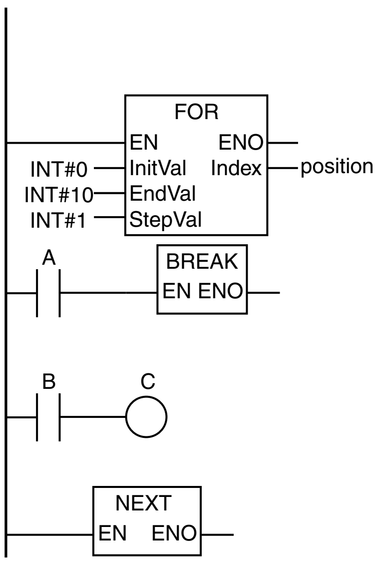
None

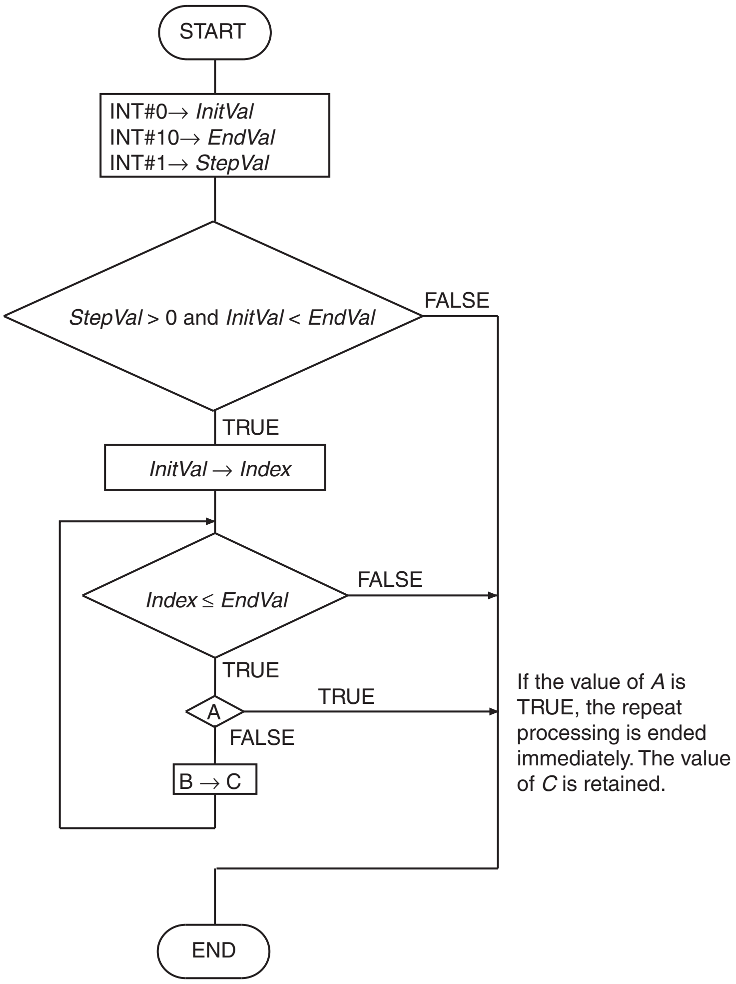
Function

The BREAK instruction cancels the repeat processing from the lowest level FOR instruction to the NEXT instruction. It moves processing to the next instruction after the NEXT instruction. The processing between the BREAK instruction and the NEXT instruction is not executed.

The following figure shows a programming example. When the FOR loop is executed, the value of variable *A* is checked each time. If the value of variable *A* is TRUE, the repeat processing is ended immediately. In this example, the Out instruction after the BREAK instruction is not executed, and the value of variable *C* is retained.

LD





Precautions for Correct Use

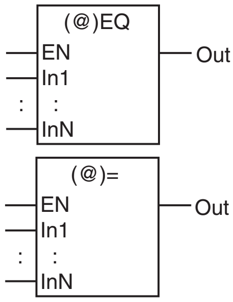
- Always place this instruction between the FOR and NEXT instructions.
- If you nest FOR and NEXT instructions, one BREAK instruction is required for each nesting level to end all of the repeat processing.
- Do not use Jump Instructions (e.g., the JMP instruction) to interrupt repeat processing. Always use a BREAK instruction to cancel repeat processing.

Comparison Instructions

Instruction	Name	Page
EQ (=)	Equal	2-92
NE (<>)	Not Equal	2-94
LT (<), LE (<=), GT (>), and GE (>=)	Less Than/Less Than Or Equal/ Greater Than/Greater Than Or Equal	2-97
EQascii	Text String Comparison Equal	2-100
NEascii	Text String Comparison Not Equal	2-102
LTascii, LEascii, GTascii, and GEascii	Text String Comparison Less Than/Text String Comparison Less Than or Equal Text String Comparison Greater Than/Text String Comparison Greater Than or Equal	2-104
Cmp	Compare	2-107
ZoneCmp	Zone Comparison	2-109
TableCmp	Table Comparison	2-111
AryCmpEQ and AryCmpNE	Array Comparison Equal/ Array Comparison Not Equal	2-114
AryCmpLT, AryCmpLE, AryCmpGT, and AryCmpGE	Array Comparison Less Than/Array Comparison Less Than Or Equal Array Comparison Greater Than/Array Comparison Greater Than Or Equal	2-116
AryCmpEQV and AryCmpNEV	Array Value Comparison Equal/Array Value Comparison Not Equal	2-119
AryCmpLTV, AryCmpLEV, AryCmpGTV, and AryCmpGEV	Array Value Comparison Less Than/Array Value Comparison Less Than Or Equal Array Value Comparison Greater Than/Array Value Comparison Greater Than Or Equal	2-121

EQ (=)

The EQ (=) instruction determines if the contents of two or more variables are all equivalent.

Instruction	Name	FB/FUN	Graphic expression	ST expression
EQ (=)	Equal	FUN		Out:=(In1=In2) & (In2=In3) & ··· & (InN-1=InN);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1 to InN	Comparison data	Input	Values to compare, N = 2 to 5	Depends on data type.	---	0*
Out	Comparison result	Output	Comparison result	Depends on data type.	---	---

* If you omit the input parameter that connects to *InN*, the default value is not applied, and a building error will occur. For example, if N is 3 and the input parameters that connect to *In1* and *In2* are omitted, the default values are applied, but if the input parameter that connects to *In3* is omitted, a building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
																*				
	Enumerations can also be specified.																			
Out	OK																			

* You can specify TIME, DATE, TOD, DT and STRING data with Sysmac Studio version 1.02 or higher. If you open a project that was created with Sysmac Studio version 1.01 or lower on Sysmac Studio version 1.02 or higher and then use any of these data types, refresh the display. To refresh the display, right-click the instruction in the Edit Pane and select ***Update***. If you do not refresh the display, a building error will occur.

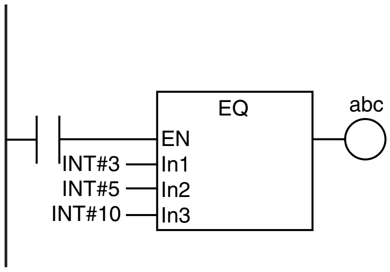
Function

The EQ (=) instruction determines if the contents of from two to five variables *In1* to *InN* are all equivalent. The comparison result *Out* is TRUE only when all values are equivalent. Otherwise, the value of *Out* is FALSE.

When comparing STRING data, “equivalent” means that both the lengths and contents of the text strings are the same.

The following example is for when *In1* is INT#3, *In2* is INT#5 and *In3* is INT#10. The value of variable *abc* will be FALSE.

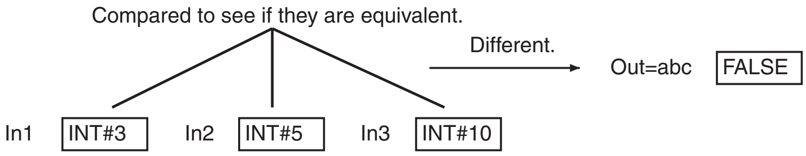
LD



ST

```
abc:=(INT#3=INT#5)&(INT#5=INT#10);
```

The EQ instruction determines if *In1* to *In3* are all equivalent.
If they are different, the value of *abc* will be FALSE.



Additional Information

- The functions of the EQ instruction and the = instruction are exactly the same. Use the form that is easier to use.
- When you compare TIME, DT, or TOD data, adjust the data so that the precision of the values is the same. Use the following instructions to adjust the precision of the values: TruncTime (page 2-657), TruncDt (page 2-661), and TruncTod (page 2-665).

Precautions for Correct Use

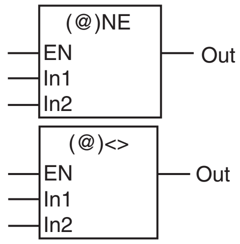
- If the data types of *In1* to *InN* are different, they will be expanded to a data type that includes the ranges of all of the data types.
- You cannot compare bit string data (BYTE, WORD, DWORD, or LWORD) with integers. You cannot compare bit string data to real number data (SINT, INT, DINT, LINT, USINT, UDINT, ULINT, REAL, and LREAL).
- Signed integers (SINT, INT, DINT, and LINT) cannot be compared to unsigned integers (USINT, UINT, UDINT, and ULINT).
- Always compare data with the same data type for TIME, DATE, TOD, DT, and STRING data. If variables with different data types are specified, a building error will occur.
- You can compare enumerations only to other enumerations. The data types must also be the same to compare enumerations.
- Two values that are positive infinity or two values that are negative infinity are equivalent.
- If any of the values of *In1* to *InN* is nonnumeric data, the value of *Out* is FALSE.
- If this instruction is used in a ladder diagram, the value of *Out* changes to FALSE if an error occurs in the previous instruction on the rung.
- If *In1* to *InN* are real numbers, the desired results may not be achieved due to rounding error. Do not use this instruction to check if two values are equal when one or more of them is a real number. Use a value comparison instruction and check to see if the difference in the absolute values is within the allowable range. For example, the following programming can be used to check to see if the sum of REAL variables *real_a* and *real_b* is equal to 0.1. If the value of BOOL variable *boolv* is TRUE, the two values are considered to be equal.

  ```
  boolv := (ABS((real_a + real_b) - 0.1) < threshold);
  ```

  threshold: Value for allowable range

NE (<>)

The NE (<>) instruction determines if the contents of two variables are not equivalent.

Instruction	Name	FB/FUN	Graphic expression	ST expression
NE (<>)	Not Equal	FUN		Out:=(In1<>In2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1 and In2	Comparison data	Input	Values to compare	Depends on data type.	---	*
Out	Comparison result	Output	Comparison result	Depends on data type.	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

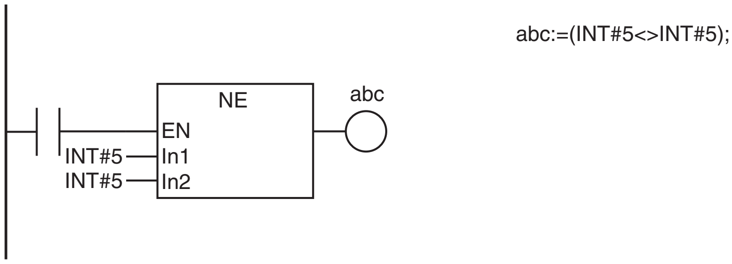
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 and In2	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
																*				
	Enumerations can also be specified.																			
Out	OK																			

* You can specify TIME, DATE, TOD, DT, and STRING data with Sysmac Studio version 1.02 or higher. If you open a project that was created with Sysmac Studio version 1.01 or lower on Sysmac Studio version 1.02 or higher and then use any of these data types, refresh the display. To refresh the display, right-click the instruction in the Edit Pane and select ***Update***. If you do not refresh the display, a building error will occur.

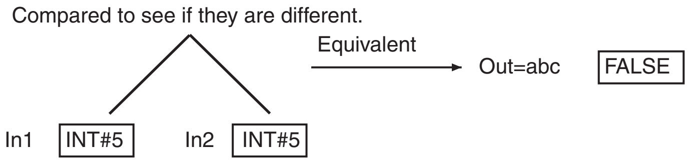
Function

The NE (<>) instruction determines if the contents of two variables *In1* and *In2* are not equivalent. If they are not equivalent, the comparison result *Out* is TRUE. If they are equivalent, *Out* is FALSE.
When comparing STRING data, “equivalent” means that both the lengths and contents of the text strings are the same.

The following example is for when *In1* equals *In2* (both have a value of INT#5). The value of variable *abc* will be FALSE.



The NE instruction determines if *In1* and *In2* are different.
If they are the same, the value of *abc* will be FALSE.



Additional Information

- The functions of the NE instruction and the <> instruction are exactly the same. Use the form that is easier to use.
- When you compare TIME, DT, or TOD data, adjust the data so that the precision of the values is the same. Use the following instructions to adjust the precision of the values: TruncTime (page 2-657), TruncDt (page 2-661), and TruncTod (page 2-665).

Precautions for Correct Use

- If the data types of *In1* and *In2* are different, the smaller one is expanded to a data type that includes the ranges of both of the data types.
- You cannot compare bit string data (BYTE, WORD, DWORD, or LWORD) with integers (SINT, INT, DINT, LINT, USINT, UDINT, ULINT). You cannot compare bit string data with real number data (REAL and LREAL).
- Signed integers (SINT, INT, DINT, and LINT) cannot be compared to unsigned integers (USINT, UINT, UDINT, and ULINT).
- Always compare data with the same data type for TIME, DATE, TOD, DT, and STRING data. If variables with different data types are specified, a building error will occur.
- You can compare enumerations only to other enumerations. The data types must also be the same to compare enumerations.
- Two values that are positive infinity or two values that are negative infinity are equivalent.
- If the value of either *In1* or *In2* is nonnumeric data, the value of *Out* is TRUE.
- If this instruction is used in a ladder diagram, the value of *Out* changes to FALSE if an error occurs in the previous instruction on the rung.

- If *In1* and *In2* are real numbers, the desired results may not be achieved due to rounding error. Do not use this instruction to check if two values are different when one or both of them is a real number. Use a value comparison instruction and check to see if the difference in the absolute values is greater than the allowable range. For example, the following programming can be used to check to see if the sum of REAL variables *real_a* and *real_b* is not equal to 0.1. If the value of BOOL variable *boolv* is TRUE, the two values are considered to be not equal.

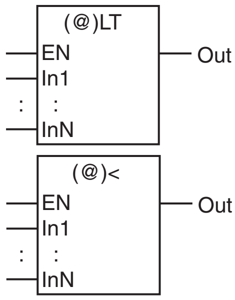
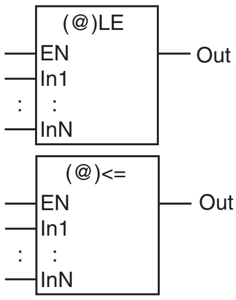
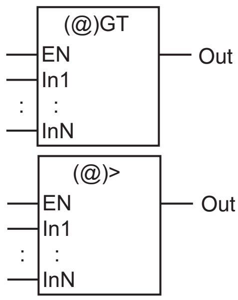
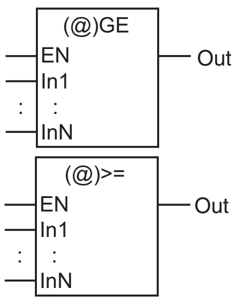
  ```
  boolv := (ABS((real_a + real_b) - 0.1) > threshold);
  ```

  threshold: Value for allowable range

LT (<), LE (<=), GT (>), and GE (>=)

These instructions compare the sizes of two or more values.

LT (<):	Performs a less than comparison.
LE (<=):	Performs a less than or equal comparison.
GT (>):	Performs a greater than comparison.
GE (>=):	Performs a greater than or equal comparison.

Instruction	Name	FB/FUN	Graphic expression	ST expression
LT (<)	Less Than	FUN		Out:=(In1<In2) & (In2<In3) & ··· & (InN-1<InN);
LE (<=)	Less Than Or Equal	FUN		Out:=(In1<=In2) & (In2<=In3) & ··· & (InN-1<=InN);
GT (>)	Greater Than	FUN		Out:=(In1>In2) & (In2>In3) & ··· & (InN-1>InN);
GE (>=)	Greater Than Or Equal	FUN		Out:=(In1>=In2) & (In2>=In3) & ··· & (InN-1>=InN);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1 to InN	Comparison data	Input	Values to compare, N = 2 to 5	Depends on data type.	---	0*
Out	Comparison result	Output	Comparison result	Depends on data type.	---	---

* If you omit the input parameter that connects to *InN*, the default value is not applied, and a building error will occur. For example, if N is 3 and the input parameters that connect to *In1* and *In2* are omitted, the default values are applied, but if the input parameter that connects to *In3* is omitted, a building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN		OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
		*														*				
Out	OK																			

* You can specify BYTE, WORD, DWORD, LWORD, TIME, DATE, TOD, DT, and STRING data with Sysmac Studio version 1.02 or higher. If you open a project that was created with Sysmac Studio version 1.01 or lower on Sysmac Studio version 1.02 or higher and then use any of these data types, refresh the display. To refresh the display, right-click the instruction in the Edit Pane and select ***Update***. If you do not refresh the display, a building error will occur.

Function

These instructions compare the data in In1 to *InN* (N = 2 to 5).

The output value *Out* is shown below for each instruction.

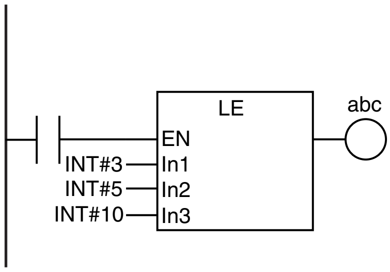
Instruction	Value of *Out*
LT (<)	If *In1* < *In2* < ... < *InN*, *Out* is TRUE. Otherwise, it is FALSE.
LE (<=)	If *In1* <= *In2* <= ... <= *InN*, *Out* is TRUE. Otherwise, it is FALSE.
GT (>)	If *In1* > *In2* > ... > *InN*, *Out* is TRUE. Otherwise, it is FALSE.
GE (>=)	If *In1* >= *In2* >= ... >= *InN*, *Out* is TRUE. Otherwise, it is FALSE.

The relationship between values with data types that are not integers or real numbers are determined as given in the following table.

Data type	Relationship
BYTE, WORD, DWORD, or LWORD	The data is compared as unsigned integers.
TIME	The numerically larger value is considered to be larger.
DATE, TOD, or DT	Later dates or times of day are considered to be larger.
STRING	The specifications are the same as for the LTascii, LEascii, GTascii, and GEascii instructions (page 2-104). Refer to the specified page for details.

The following example shows the LE instruction when *In1* is INT#3, *In2* is INT#5 and *In3* is INT#10. The value of variable *abc* will be TRUE.

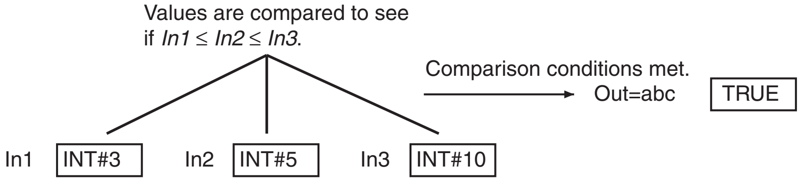
LD



ST

```
abc:=(INT#3<= INT#5)&(INT#5<=INT#10);
```

The LE instruction determines if *In1* ≤ *In2* ≤ *In3*.
If the comparison conditions are met, the value of *abc* will be TRUE.



Additional Information

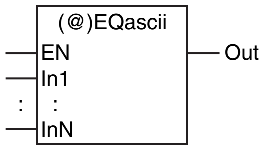
- The functions of the LT and < instructions, the LE and <= instructions, the GT and > instructions, and the GE and >= instructions are exactly the same. Use the form that is easier to use.
- When you compare TIME, DT, or TOD data, adjust the data so that the precision of the values is the same. Use the following instructions to adjust the precision of the values: TruncTime (page 2-657), TruncDt (page 2-661), and TruncTod (page 2-665).

Precautions for Correct Use

- If the data types of *In1* to *InN* are different, they will be expanded to a data type that includes the ranges of all of the data types.
- Signed integers (SINT, INT, DINT, and LINT) cannot be compared to unsigned integers (USINT, UINT, UDINT, and ULINT).
- You cannot compare bit string data (BYTE, WORD, DWORD, or LWORD) with integers (SINT, INT, DINT, LINT, USINT, UINT, UDINT, or ULINT). You cannot compare bit string data with real number data (REAL or LREAL).
- Always compare data with the same data type for TIME, DATE, TOD, DT, and STRING data. If variables with different data types are specified, a building error will occur.
- If *In1* to *InN2* are real numbers, error may cause unexpected processing results. This can occur, for example, when they contain non-terminating decimal numbers.
- Two values that are positive infinity or two values that are negative infinity are equivalent.
- If any of the values of *In1* to *InN* is nonnumeric data, the value of *Out* is FALSE.
- If this instruction is used in a ladder diagram, the value of *Out* changes to FALSE if an error occurs in the previous instruction on the rung.

EQascii

The EQascii instruction determines if two or more text strings are all equivalent.

Instruction	Name	FB/FUN	Graphic expression	ST expression
EQascii	Text String Comparison Equal	FUN		Out:=EQascii(In1, ··, InN);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1 to InN	Comparison text strings	Input	Text strings to compare, N = 2 to 5	Depends on data type.	---	'' *
Out	Comparison result	Output	Comparison result	Depends on data type.	---	---

* If you omit the input parameter that connects to *InN*, the default value is not applied, and a building error will occur. For example, if N is 3 and the input parameters that connect to *In1* and *In2* are omitted, the default values are applied, but if the input parameter that connects to *In3* is omitted, a building error will occur.

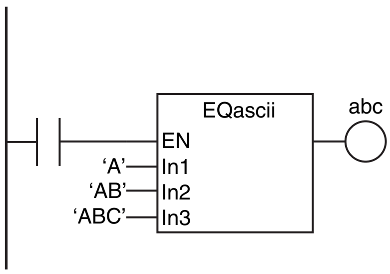
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN																				OK
Out	OK																			

Function

The EQascii instruction determines if from two to five text strings *In1* to *InN* are all equivalent. If the are all equivalent, comparison result *Out* changes to TRUE. Otherwise, the value of *Out* is FALSE. "Equivalent" means that both the lengths and contents of the text strings are the same.

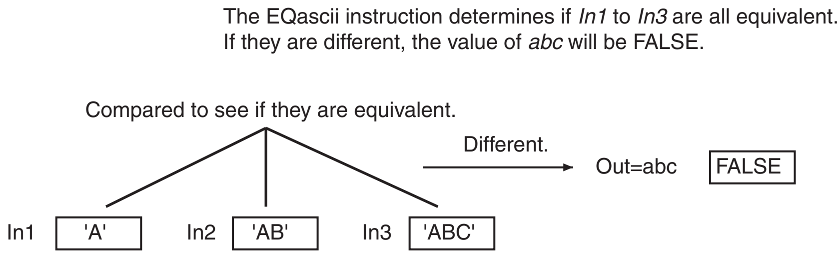
The following example is for when *In1* is "A", *In2* is "AB", and *In3* is "ABC". The value of variable *abc* will be FALSE.

LD



ST

abc:=EQascii('A', 'AB', 'ABC');



Additional Information

The text string comparison instructions are convenient when you want to reorder text strings according to the character codes. For example, the character codes for alphabet characters are in the same order as the alphabet characters. This allows you to alphabetize.

Version Information

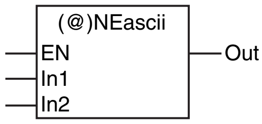
With Sysmac Studio version 1.02 or higher, the EQ (=) instruction (page 2-92) can also be used to compare text strings. The specifications of the EQ (=) instruction for comparing text strings are the same as for the EQascii instruction.

Precautions for Correct Use

- Do not use this instruction as the rightmost instruction on a rung. If you do, an error occurs on the Sysmac Studio and you cannot transfer the user program to the Controller.
- If this instruction is used in a ladder diagram, the value of *Out* changes to FALSE if an error occurs in the previous instruction on the rung.
- Specify text strings that contain only ASCII characters for *In1* to *InN*.

NEascii

The NEascii instruction determines if two text strings are not equivalent.

Instruction	Name	FB/FUN	Graphic expression	ST expression
NEascii	Text String Comparison Not Equal	FUN		Out:=NEascii(In1, In2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1 and In2	Comparison text strings	Input	Text strings to compare	Depends on data type.	---	*
Out	Comparison result	Output	Comparison result	Depends on data type.	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

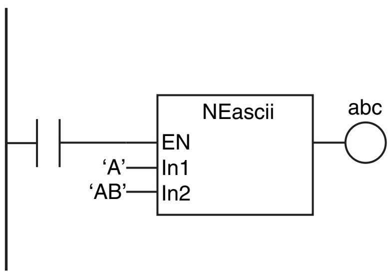
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 and In2																				OK
Out	OK																			

Function

The NEascii instruction determines if two text strings *In1* and *In2* are not equivalent. If they are different, comparison result *Out* will be TRUE. If they are the same, comparison result *Out* will be FALSE. “Equivalent” means that both the lengths and contents of the text strings are the same.

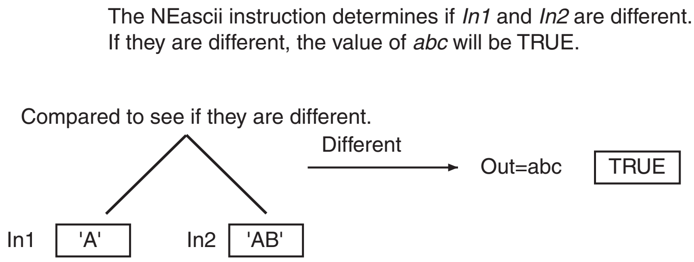
The following example is for when *In1* is “A” and *In2* is “AB”. The value of variable *abc* will be TRUE.

LD



ST

```
abc:=NEascii('A', 'AB');
```



Additional Information

The text string comparison instructions are convenient when you want to reorder text strings according to the character codes. For example, the character codes for alphabet characters are in the same order as the alphabet characters. This allows you to alphabetize.

Version Information

With Sysmac Studio version 1.02 or higher, the NE (<>) instruction (page 2-94) can also be used to compare text strings. The specifications of the NE (<>) instruction for comparing text strings are the same as for the NEascii instruction.

Precautions for Correct Use

- Do not use this instruction as the rightmost instruction on a rung. If you do, an error occurs on the Sysmac Studio and you cannot transfer the user program to the Controller.
- If this instruction is used in a ladder diagram, the value of *Out* changes to FALSE if an error occurs in the previous instruction on the rung.
- Specify text strings that contain only ASCII characters for *In1* and *In2*.

LTascii, LEascii, GTascii, and GEascii

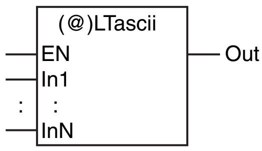
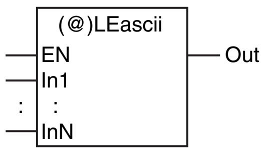
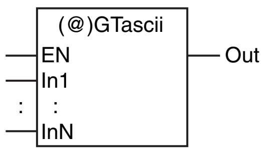
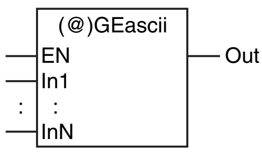
These instructions compare the sizes of two or more text strings.

LTascii: Performs a less than comparison.

LEascii: Performs a less than or equal comparison.

GTascii: Performs a greater than comparison.

GEascii: Performs a greater than or equal comparison.

Instruction	Name	FB/FUN	Graphic expression	ST expression
LTascii	Text String Comparison Less Than	FUN		Out:=LTascii(In1, ···, InN);
LEascii	Text String Comparison Less Than or Equal	FUN		Out:=LEascii(In1, ···, InN);
GTascii	Text String Comparison Greater Than	FUN		Out:=GTascii(In1, ···, InN);
GEascii	Text String Comparison Greater Than or Equal	FUN		Out:=GEascii(In1, ···, InN);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1 to InN	Comparison text strings	Input	Text strings to compare, N = 2 to 5	Depends on data type.	---	''*
Out	Comparison result	Output	Comparison result	Depends on data type.	---	---

* If you omit the input parameter that connects to *InN*, the default value is not applied, and a building error will occur. For example, if N is 3 and the input parameters that connect to *In1* and *In2* are omitted, the default values are applied, but if the input parameter that connects to *In3* is omitted, a building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN																				OK
Out	OK																			

Function

These instructions compare the sizes of from two to five text strings in *In1* to *InN* (N = 2 to 5). The output value *Out* is shown below for each instruction.

Instruction	Value of *Out*
LTascii	If *In1* < *In2* <...< *InN*, *Out* is TRUE. Otherwise, it is FALSE.
LEascii	If *In1* ≤ *In2* ≤...≤ *InN*, *Out* is TRUE. Otherwise, it is FALSE.
GTascii	If *In1* > *In2* >...> *InN*, *Out* is TRUE. Otherwise, it is FALSE.
GEascii	If *In1* ≥ *In2* ≥...≥ *InN*, *Out* is TRUE. Otherwise, it is FALSE.

The sizes of the character codes are compared. The comparison procedure is as follows:

First, the first character codes in all of the text strings are compared. If the character codes are different, the result of the size comparison for the text strings is determined by the size relationship between those character codes. If the character codes are the same, comparison continues in order to the other characters until a different character code is found. If the lengths of the text strings are different, NULL characters (16#00) are added to the shorter text string to complete the comparison.

The relationships between various text strings are as follows:

'AD'(16#414400) < 'BC'(16#424400)

'ADC' (16#41444300)< 'B'(16#42000000)

'ABC' (16#41424300)< 'ABD'(16#41424400)

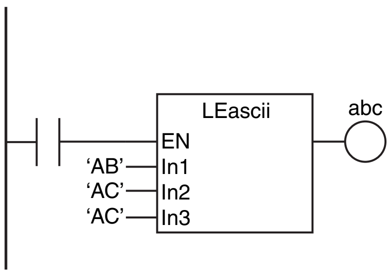
'ABC' (16#41424300)> 'AB'(16#41420000)

'AB' (16#414200)= 'AB'(16#414200)

If the text string contains multi-byte characters, the characters are separated into individual bytes before comparison. For example, the two-byte character 16#C281 is handled as 16#C2 and 16#81.

The following example for the LEascii instruction is for when *In1* is “AB”, *In2* is “AC”, and *In3* is “AC”. The value of variable *abc* will be TRUE.

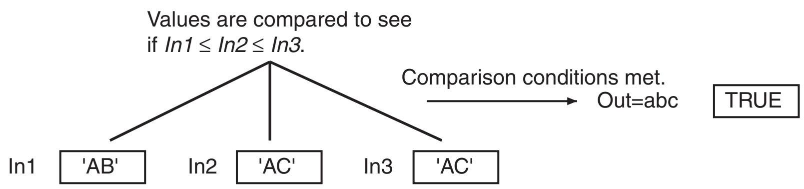
LD



ST

```
abc:=LEascii(’AB’, ‘AC’, ‘AC’);
```

The LEascii instruction determines if *In1* ≤ *In2* ≤ *In3*.
If the comparison conditions are met, the value of *abc* will be TRUE.



Additional Information

The text string comparison instructions are convenient when you want to reorder text strings according to the character codes. For example, the character codes for alphabet characters are in the same order as the alphabet characters. This allows you to alphabetize.

Version Information

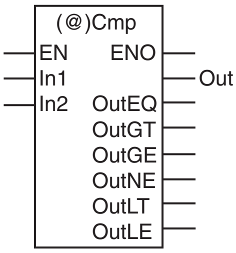
With Sysmac Studio version 1.02 or higher, the LT (<), LE (<=), GT (>), and GE (>=) instructions (page 2-97) can also be used to compare text strings. The specifications of the LT (<), LE (<=), GT (>), and GE (>=) instructions for comparing text strings are the same as for the LTascii, LEascii, GTascii, and GEascii instructions.

Precautions for Correct Use

- Do not use this instruction as the rightmost instruction on a rung. If you do, an error occurs on the Sysmac Studio and you cannot transfer the user program to the Controller.
- If this instruction is used in a ladder diagram, the value of *Out* changes to FALSE if an error occurs in the previous instruction on the rung.
- Specify text strings that contain only ASCII characters for *In1* to *InN*.

Cmp

The Cmp instruction compares two values.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Cmp	Compare	FUN		Out:=Cmp(In1, In2, OutEQ, OutGT, OutGE, OutNE, OutLT, OutLE); You can omit *Out*.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1 and In2	Comparison data	Input	Values to compare	Depends on data type.	---	*
Out	Return value	Output	Always TRUE	TRUE only	---	---
OutEQ	Equal flag		Equal flag	Depends on data type.		
OutGT	Greater than flag		Greater than flag			
OutGE	Greater than or equal flag		Greater than or equal flag			
OutNE	Not equal flag		Not equal flag			
OutLT	Less than flag		Less than flag			
OutLE	Less than or equal flag		Less than or equal flag			

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 and In2						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Out	OK																			
OutEQ	OK																			
OutGT	OK																			
OutGE	OK																			
OutNE	OK																			
OutLT	OK																			
OutLE	OK																			

Function

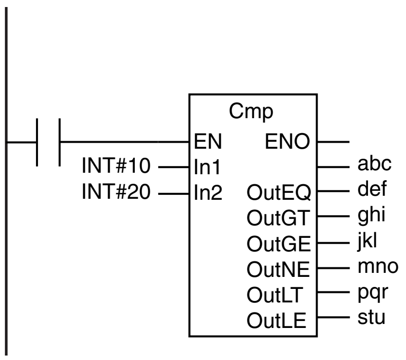
The Cmp instruction compares two values (*In1* and *In2*) and outputs flag values.

The values of the flags are as follows:

Flag	Value
OutEQ	If *In1* equals *In2*, the flag shows TRUE. Otherwise the flag shows FALSE.
OutGT	If *In1* is greater than *In2*, the flag shows TRUE. Otherwise the flag shows FALSE.
OutGE	If *In1* is greater than or equal to *In2*, the flag shows TRUE. Otherwise the flag shows FALSE.
OutNE	If *In1* is not equal to *In2*, the flag shows TRUE. Otherwise the flag shows FALSE.
OutLT	If *In1* is less than *In2*, the flag shows TRUE. Otherwise the flag shows FALSE.
OutLE	If *In1* is less than or equal to *In2*, the flag shows TRUE. Otherwise the flag shows FALSE.

The following example is for when *In1* is INT#10 and *In2* is INT#20. The values of variables *def*, *ghi*, and *jkl* will be FALSE, and the values of *abc*, *mno, pqr*, and *stu* will be TRUE.

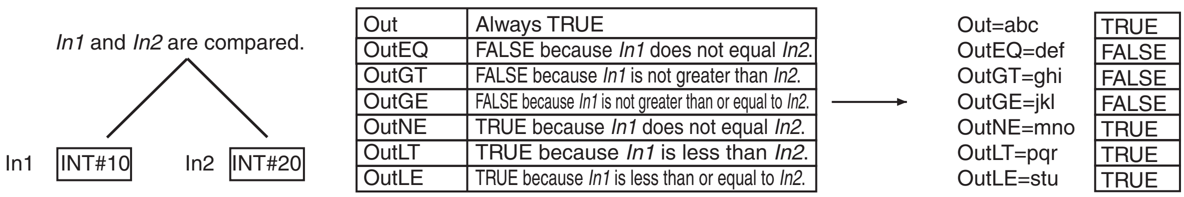
LD



ST

```
abc:=Cmp(INT#10, INT#20, def, ghi, jkl, mno, pqr, stu);
```

The Cmp instruction compares *In1* and *In2*.
The results are given below for the various criteria.



Precautions for Correct Use

- If the data types of *In1* and *In2* are different, the smaller one is expanded to a data type that includes the ranges of both of the data types.
- If *In1* and *In2* are real numbers, error may cause unexpected processing results. This can occur, for example, when they contain non-terminating decimal numbers.
- Signed integers (SINT, INT, DINT, and LINT) cannot be compared to unsigned integers (USINT, UINT, UDINT, and ULINT).
- Two values that are positive infinity or two values that are negative infinity are equivalent.
- If the value of either *In1* or *In2* is nonnumeric data, the values of *OutEQ*, *OutGT*, *OutGE*, *OutNE*, *OutLT*, and *OutLE* are FALSE.

ZoneCmp

The ZoneCmp instruction determines if the comparison data is within the specified maximum and minimum values.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ZoneCmp	Zone Comparison	FUN	(@)ZoneCmp EN — Out MN In MX	Out:=ZoneCmp(MN, In, MX);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
MN	Minimum value	Input	Minimum value	Depends on data type.	---	0
In	Comparison data		Value to compare			*
MX	Maximum value		Maximum value			0
Out	Comparison result	Output	Comparison result	Depends on data type.	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
MN						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	
																*				
In						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	
																*				
MX						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	
																*				
Out	OK																			

* You can specify TIME, DATE, TOD, and DT data with CPU Units with unit version 1.01 or later and Sysmac Studio version 1.02 or higher.

Function

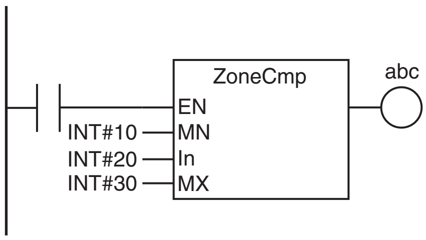
The ZoneCmp instruction determines if comparison data *In* is between maximum value *MX* and minimum value *MN*. If MX ≥ In ≥ MN, *Out* will be TRUE. Otherwise, *Out* will be FALSE.

The relationship between values with data types that are not integers or real numbers are determined as given in the following table.

Data type	Relationship
TIME	The numerically larger value is considered to be larger.
DATE, TOD, or DT	Later dates or times of day are considered to be larger.

The following example is for when *MN* is INT#10, *In* is INT#20 and *MX* is INT#30. The value of variable *abc* will be TRUE.

LD

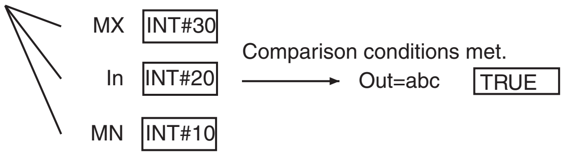


ST

```
abc:=ZoneCmp(INT#10, INT#20, INT#30);
```

The ZoneCmp instruction determines if *MX* ≥ *In* ≥ *MN*.
If the comparison conditions are met, the value of *abc* will be TRUE.

The instruction determines if *In* is between *MX* and *MN*.



Additional Information

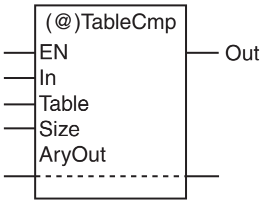
When you compare TIME, DT, or TOD data, adjust the data so that the precision of the values is the same. Use the following instructions to adjust the precision of the values: TruncTime (page 2-657), TruncDt (page 2-661), and TruncTod (page 2-665).

Precautions for Correct Use

- If the data types of *In, MX,* and *MN* are different, they will be expanded to a data type that includes the ranges of all of the data types.
- If *In*, *MX*, and *MN* are real numbers, error may cause unexpected processing results. This can occur, for example, when they contain non-terminating decimal numbers.
- Signed integers (SINT, INT, DINT, and LINT) cannot be compared to unsigned integers (USINT, UINT, UDINT, and ULINT).
- Always compare data with the same data type for TIME, DATE, TOD, and DT data. If variables with different data types are specified, a building error will occur.
- Two values that are positive infinity or two values that are negative infinity are equivalent.
- If the value of *In* is nonnumeric data, the value of *Out* is FALSE.
- If this instruction is used in a ladder diagram, the value of *Out* changes to FALSE if an error occurs in the previous instruction on the rung.
- An error occurs in the following cases. *Out* will be FALSE.
  - The value of *MN* is greater than the value of *MX.*
  - Either *MX* or *MN* contains nonnumeric data.

TableCmp

The TableCmp instruction compares the comparison data with multiple defined ranges in a comparison table.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TableCmp	Table Comparison	FUN		Out:=TableCmp(In, Table, Size, AryOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Comparison data	Input	Value to compare	Depends on data type.	---	*
Table[] (two-dimensional array)	Comparison table		Two-dimensional array that contains the elements for the defined ranges			
Size	Comparison size		Number of elements in *Table[]* to which to compare *In*			1
AryOut[] (array)	Individual comparison results array	In-out	Comparison results for *Table[]* elements TRUE: Condition met. FALSE: Condition not met.	Depends on data type.	---	---
Out	Comparison result	Output	TRUE: *In* meets all comparison conditions for elements of *Table[]*. FALSE: The comparison condition is not met for one or more sets of elements.	Depends on data type.	---	---

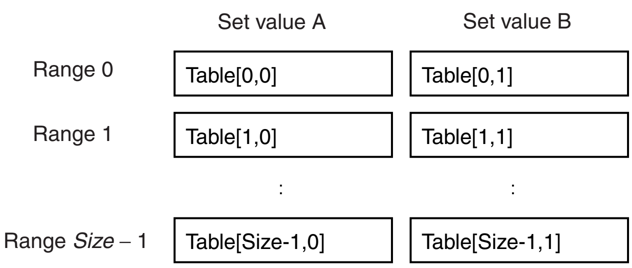
* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Table[] (two-dimensional array)	Must be a two-dimensional array with elements that have the same data type as *In*.																			
Size							OK													
AryOut[] (array)	OK																			
Out	OK																			

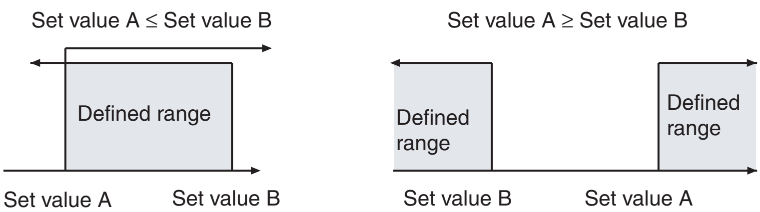
Function

The TableCmp instruction compares comparison data *In* with the number of defined ranges specified by the value of *Size* in comparison table *Table[]*.

Table[] is a two-dimensional array. The first dimension contains the numbers of the defined ranges. In the second dimension, element 0 is set value A of the defined range and element 1 is set value B of the defined range.



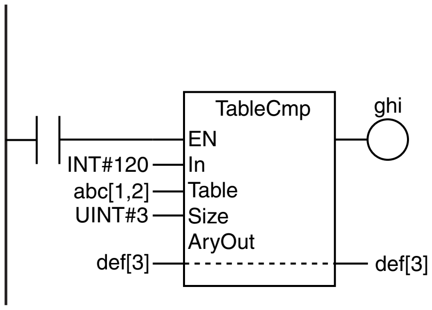
Set value A and set value B define range as shown below. Set value A and set value B are always included in the range.



The results of comparing *In* and *Table[]* are stored in individual comparison results array *AryOut[]*. If *In* is within the defined range for element *i*, *AryOut[i]* will be TRUE. If it is not within the range, *AryOut[i]* will be FALSE. If all *Size* elements of *AryOut[]* are TRUE, comparison result *Out* will be TRUE. Otherwise, it will be FALSE.

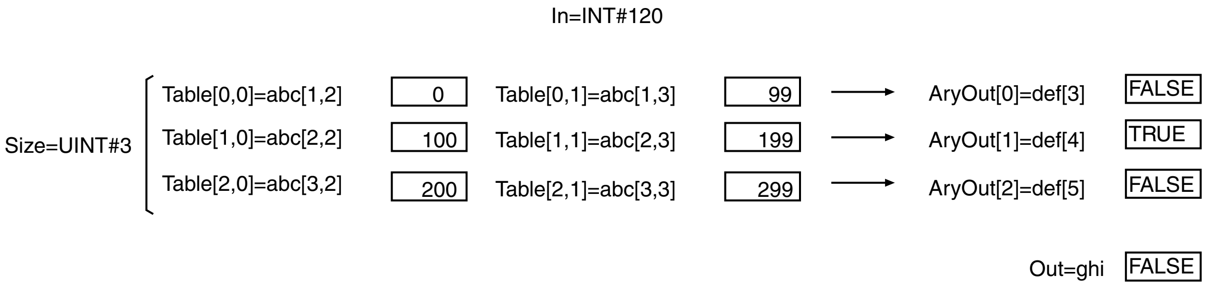
The following example is for when *In* is INT#120 and *Size* is UINT#3.

LD



ST

```
ghi:=TableCmp(INT#120, abc[1,2], UINT#3, def[3]);
```



Precautions for Correct Use

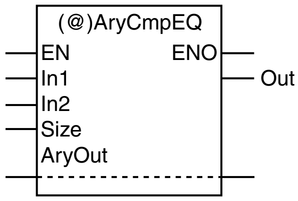
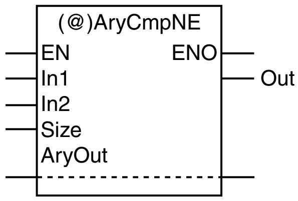
- Use the same data type for *In* and *Table[].* Otherwise, a building error will occur.
- Use a two-dimensional array for *Table[].*
- If an array with more than two dimensions is used for *Table[]*, the elements in the third and higher dimensions are ignored.
- If the *AryOut[]* array is larger than the value of *Size*, the comparison results will be stored in *AryOut[0]* to *AryOut[Size−1]*. Other elements of the array will not change.
- Signed integers (SINT, INT, DINT, and LINT) cannot be compared to unsigned integers (USINT, UINT, UDINT, and ULINT).
- If real numbers are compared, error may cause unexpected processing results. This can occur, for example, when they contain non-terminating decimal numbers.
- If the value of *Size* is 0, the value of *Out* will be FALSE and *AryOut[]* will not change.
- If this instruction is used in a ladder diagram, the value of *Out* changes to FALSE if an error occurs in the previous instruction on the rung.
- An error occurs in the following cases. *Out* will be FALSE.
    - If the value of *Size* exceeds the size of the *AryOut[]* array.
    - If the value of *Size* exceeds the size of the first dimension of the *Table[]* array.
    - The size of the second dimension of *Table []* is 1.

AryCmpEQ and AryCmpNE

These instructions compare the values of the elements of two arrays.

AryCmpEQ: Determines if the elements are equal.

AryCmpNE: Determines if the elements are not equal.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryCmpEQ	Array Comparison Equal	FUN		AryCmpEQ(In1, In2, Size, AryOut);
AryCmpNE	Array Comparison Not Equal	FUN		AryCmpNE(In1, In2, Size, AryOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1[] and In2[] (arrays)	Comparison arrays	Input	Arrays containing the elements to compare	Depends on data type.	---	*
Size	Number of comparison elements		Number of elements to compare	Depends on data type.		1
AryOut[] (array)	Comparison results array	In-out	Comparison results array	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[] (array)	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
In2[] (array)	Must be an array with the same data type as *In1[]*.																			
Size							OK													
AryOut[] (array)	OK																			
Out	OK																			

Function

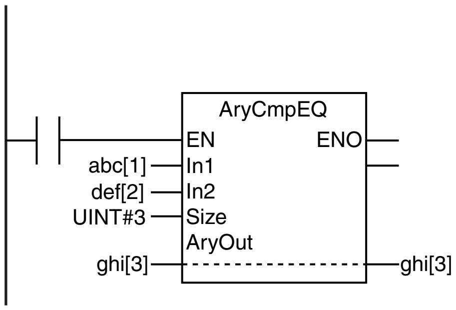
These instructions compare the values of the elements with the same element numbers in two arrays (In1[0] to In1[*Size*–1] and In2[0] and In2[*Size*–1]). The comparison results are stored in comparison results array *AryOut[]* in the elements with the corresponding element numbers (AryOut[0] to AryOut[*Size* –1]).

The value of *AryOut[i]* is as follows for each instruction:

Instruction	Value of *AryOut[i]*
AryCmpEQ	If *In1[i]* = *In2[i]*, the result is TRUE. Otherwise, it is FALSE.
AryCmpNE	If *In1[i]* ≠ *In2[i]*, the result is TRUE. Otherwise, it is FALSE.

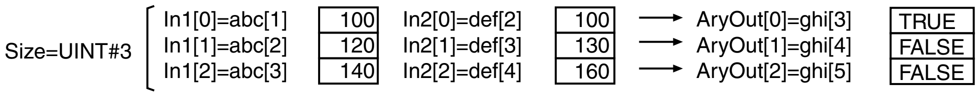
The following example shows the AryCmpEQ instruction when *Size* is UINT#3.

LD



ST

```
AryCmpEQ(abc[1], def[2], UINT#3, ghi[3]);
```



Precautions for Correct Use

- Use the same data type for *In1[]* and *In2[]*. If they are different, a building error will occur.
- Use an *AryOut[]* array that is at least as large as the value of *Size.*
- If *In1[]* and *In2[]* contain real numbers, error may cause unexpected processing results. This can occur, for example, when they contain non-terminating decimal numbers.
- If the value of *Size* is 0, the value of *Out* will be TRUE and *AryOut[]* will not change.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *AryOut[]* will not change.
  - If the *In1[]*, *In2[]*, or *AryOut[]* array is smaller than the value of *Size.*

AryCmpLT, AryCmpLE, AryCmpGT, and AryCmpGE

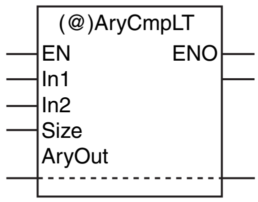
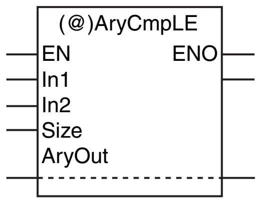
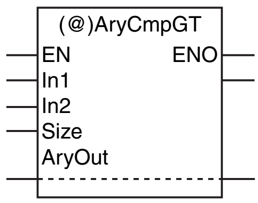
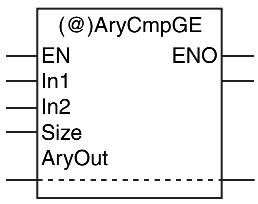
These instructions compare the values of the elements of two arrays.

AryCmpLT: Performs a less than comparison.

AryCmpLE: Performs a less than or equal comparison.

AryCmpGT: Performs a greater than comparison.

AryCmpGE: Performs a greater than or equal comparison.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryCmpLT	Array Comparison Less Than	FUN		AryCmpLT(In1, In2, Size, AryOut);
AryCmpLE	Array Comparison Less Than Or Equal	FUN		AryCmpLE(In1, In2, Size, AryOut);
AryCmpGT	Array Comparison Greater Than	FUN		AryCmpGT(In1, In2, Size, AryOut);
AryCmpGE	Array Comparison Greater Than Or Equal	FUN		AryCmpGE(In1, In2, Size, AryOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1[] and In2[] (arrays)	Comparison arrays	Input	Arrays containing the elements to compare	Depends on data type.	---	*
Size	Number of comparison elements		Number of elements to compare	Depends on data type.		1
AryOut[] (array)	Comparison results array	In-out	Comparison results array	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[] (array)						OK	OK	OK	OK	OK	OK	OK	OK	OK						
In2[] (array)	Must be an array with the same data type as *In1[]*.																			
Size							OK													
AryOut[] (array)	OK																			
Out	OK																			

Function

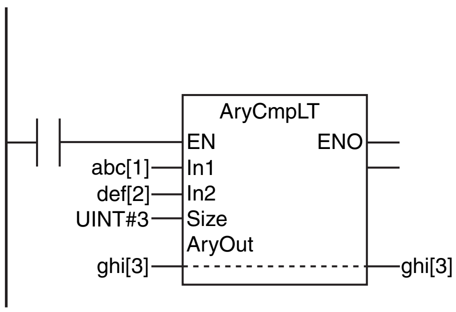
These instructions compare the values of the elements with the same element numbers in two arrays (In1[0] to In1[*Size* −1] and In2[0] and In2[*Size* −1]). The comparison results are stored in comparison results array *AryOut[]* in the elements with the corresponding element numbers (AryOut[0] to AryOut[*Size* −1]).

The value of *AryOut[i]* is as follows for each instruction:

Instruction	Value of *AryOut[i]*
AryCmpLT	If *In1[i]* < *In2[i]*, the result is TRUE. Otherwise, it is FALSE.
AryCmpLE	If *In1[i]* <= *In2[i]*, the result is TRUE. Otherwise, it is FALSE.
AryCmpGT	If *In1[i]* > *In2[i]*, the result is TRUE. Otherwise, it is FALSE.
AryCmpGE	If *In1[i]* >= *In2[i]*, the result is TRUE. Otherwise, it is FALSE.

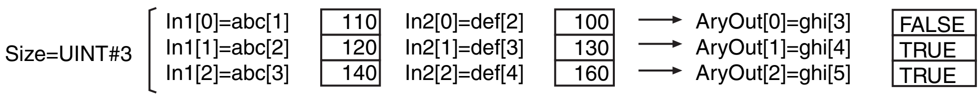
The following example shows the AryCmpLT instruction when *Size* is UINT#3.

LD



ST

```
AryCmpLT(abc[1], def[2], UINT#3, ghi[3]);
```



Precautions for Correct Use

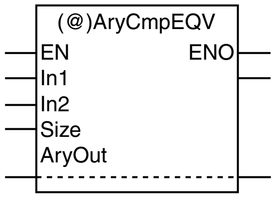
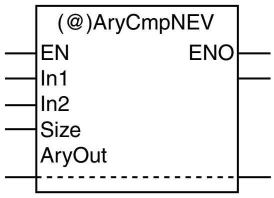
- Use the same data type for *In1[]* and *In2[].* If they are different, a building error will occur.
- Use an *AryOut[]* array that is at least as large as the value of *Size.*
- If *In1[]* and *In2[]* contain real numbers, error may cause unexpected processing results. This can occur, for example, when they contain non-terminating decimal numbers.
- If the value of *Size* is 0, the value of *Out* will be TRUE and *AryOut[]* will not change.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *AryOut[]* will not change.
  - If the *In1[]*, *In2[]*, or *AryOut[]* array is smaller than the value of *Size.*

AryCmpEQV and AryCmpNEV

These instructions compare a value to the values of the elements of an array.

AryCmpEQV: Determines if the elements are equal.

AryCmpNEV: Determines if the elements are not equal.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryCmpEQV	Array Value Comparison Equal	FUN		AryCmpEQV(In1, In2, Size, AryOut);
AryCmpNEV	Array Value Comparison Not Equal	FUN		AryCmpNEV(In1, In2, Size, AryOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1[] (array)	Comparison array	Input	Array containing the elements to compare	Depends on data type.	---	*
In2	Comparison value	Input	Value to compare	Depends on data type.	---	*
Size	Number of comparison elements	Input	Number of elements to compare	Depends on data type.	---	1
AryOut[] (array)	Comparison results array	In-out	Comparison results array	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[] (array)	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
In2	Must be same data type as the elements of *In1[]*.																			
Size							OK													
AryOut[] (array)	OK																			
Out	OK																			

Function

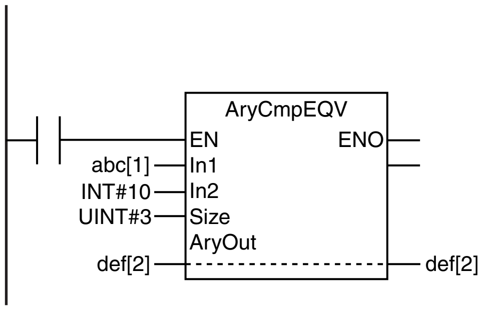
These instructions compare comparison value *In2* with the specified elements in an array (*In1*[0] to *In1*[*Size* −1]). The comparison results are stored in comparison results array *AryOut[]* in the elements with the corresponding element numbers (*AryOut*[0] to *AryOut*[*Size* −1]).

The value of *AryOut[i]* is as follows for each instruction:

Instruction	Value of *AryOut[i]*
AryCmpEQV	If *In1[i]* = *In2*, the result is TRUE. Otherwise, it is FALSE.
AryCmpNEV	If *In1[i]* ≠ *In2*, the result is TRUE. Otherwise, it is FALSE.

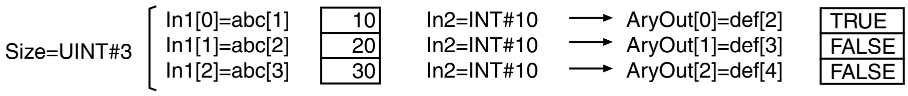
The following example shows the AryCmpEQV instruction when *In2* is INT#10 and *Size* is UINT#3.

LD



ST

```
AryCmpEQV(abc[1], INT#10, UINT#3, def[2]);
```



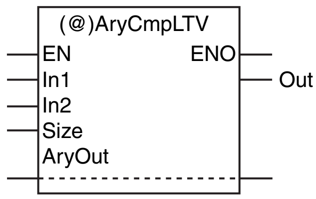
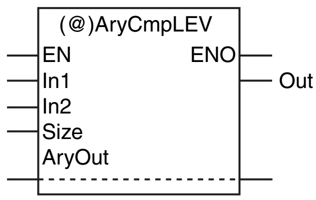
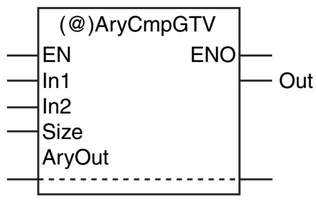
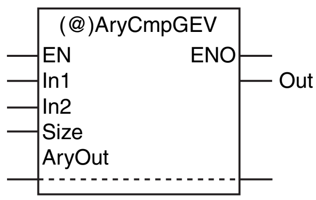
Precautions for Correct Use

- Use the same data type for *In1[]* and *In2.* If they are different, a building error will occur.
- Use an *AryOut[]* array that is at least as large as the value of *Size.*
- If *In1[]* contains real numbers and *In2* is a real number, error may cause unexpected processing results. This can occur, for example, when they contain non-terminating decimal numbers.
- If the value of *Size* is 0, the value of *Out* will be TRUE and *AryOut[]* will not change.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following case. *ENO* will be FALSE, and *AryOut[]* will not change.
  - If the *In1[]* or *AryOut[]* array is smaller than the value of *Size.*

AryCmpLTV, AryCmpLEV, AryCmpGTV, and AryCmpGEV

These instructions compare a value to the values of the elements of an array.

AryCmpLTV:	Performs a less than comparison.
AryCmpLEV:	Performs a less than or equal comparison.
AryCmpGTV:	Performs a greater than comparison.
AryCmpGEV:	Performs a greater than or equal comparison.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryCmpLTV	Array Value Comparison Less Than	FUN		AryCmpLTV(In1, In2, Size, AryOut);
AryCmpLEV	Array Value Comparison Less Than Or Equal	FUN		AryCmpLEV(In1, In2, Size, AryOut);
AryCmpGTV	Array Value Comparison Greater Than	FUN		AryCmpGTV(In1, In2, Size, AryOut);
AryCmpGEV	Array Value Comparison Greater Than Or Equal	FUN		AryCmpGEV(In1, In2, Size, AryOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1[] (array)	Comparison array	Input	Array containing the elements to compare	Depends on data type.	---	*
In2	Comparison value		Value to compare			
Size	Number of comparison elements		Number of elements to compare	Depends on data type.		1
AryOut[] (array)	Comparison results array	In-out	Comparison results array	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[] (array)						OK	OK	OK	OK	OK	OK	OK	OK	OK						
In2	Must be same data type as the elements of *In1[]*.																			
Size							OK													
AryOut[] (array)	OK																			
Out	OK																			

Function

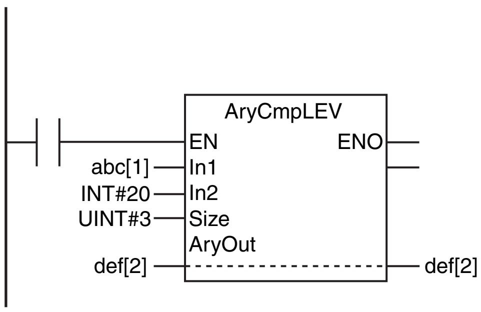
These instructions compare comparison value *In2* with the specified elements in an array (In1[0] to In1[*Size* – 1]). The comparison results are stored in comparison results array *AryOut[]* in the elements with the corresponding element numbers (AryOut[0] to AryOut[*Size* –1]).

The value of *AryOut[i]* is as follows for each instruction:

Instruction	Value of *AryOut[i]*
AryCmpLTV	If *In1[i]* < *In2*, the result is TRUE. Otherwise, it is FALSE.
AryCmpLEV	If *In1[i]* <= *In2*, the result is TRUE. Otherwise, it is FALSE.
AryCmpGTV	If *In1[i]* > *In2*, the result is TRUE. Otherwise, it is FALSE.
AryCmpGEV	If *In1[i]* >= *In2*, the result is TRUE. Otherwise, it is FALSE.

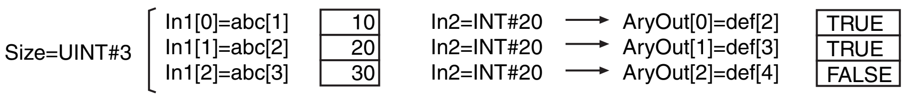
The following example shows the AryCmpLEV instruction when *In2* is INT#20 and *Size* is UINT#3.

LD



ST

```
AryCmpLEV(abc[1], INT#20, UINT#3, def[2]);
```



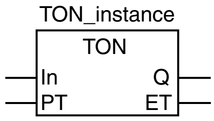
Precautions for Correct Use

- Use the same data type for *In1[]* and *In2.* If they are different, a building error will occur.
- Use an *AryOut[]* array that is at least as large as the value of *Size.*
- If *In1[]* contains real numbers and *In2* is a real number, error may cause unexpected processing results. This can occur, for example, when they contain non-terminating decimal numbers.
- If the value of *Size* is 0, the value of *Out* will be TRUE and *AryOut[]* will not change.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following case. *ENO* will be FALSE, and *AryOut[]* will not change.
  - If the *In1[]* or *AryOut[]* array is smaller than the value of *Size.*

Timer Instructions

TON

The TON instruction outputs TRUE when the set time elapses after the timer starts.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TON	On-Delay Timer	FB		TON_instance (In, PT, Q, ET);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Timer input	Input	TRUE: Timer start signal FALSE: Timer reset signal	Depends on data type.	---	FALSE
PT	Set time		Time from when timer starts until *Q* changes to TRUE	*	ms	0
Q	Timer output	Output	TRUE: Timer output ON FALSE: Timer output OFF	Depends on data type.	---	---
ET	Elapsed time		Elapsed time since timer started	*	ms	

* T#0ms to T#106751d_23h_47m_16s_854.775807ms

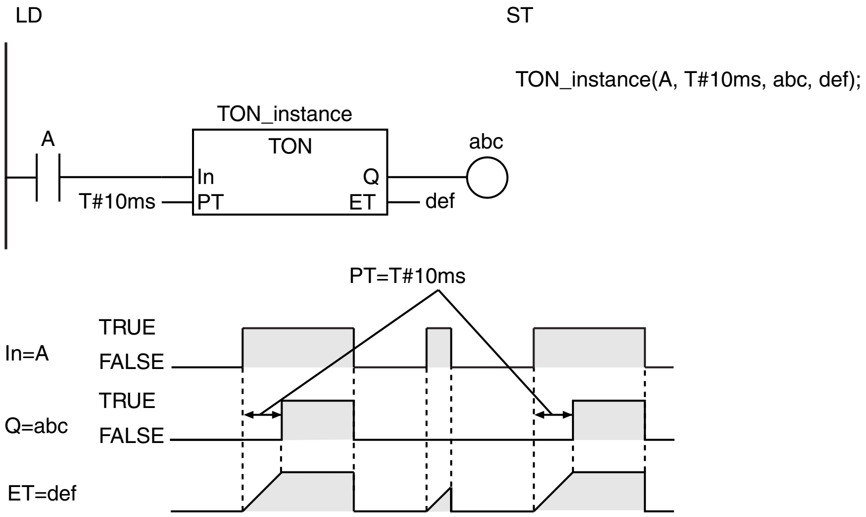
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	OK																			
PT																OK				
Q	OK																			
ET																OK				

Function

The TON instruction outputs TRUE when the set time elapses after the timer starts. The time is set in nanoseconds. The timer starts when timer input *In* changes to TRUE. Elapsed time *ET* is incremented as time elapses. When *ET* reaches set time *PT*, timer output *Q* changes to TRUE. *ET* is not incremented after that. The timer is reset when *In* changes to FALSE. *ET* changes to 0 and *Q* changes to FALSE.

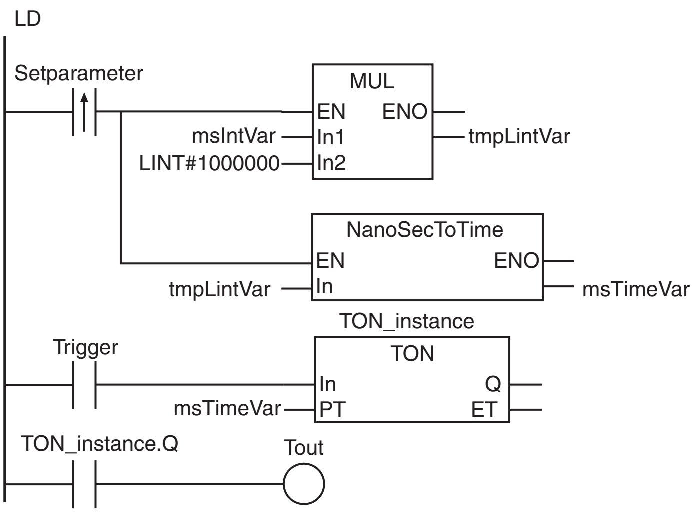
If the timer is started and then *In* changes to FALSE before *ET* reaches *PT*, the timer is reset.

The following figure shows a programming example and timing chart when *PT* is T#10ms. Variable *abc* will change to TRUE 10 ms after variable *A* changes to TRUE.



Additional Information

- Use the TP instruction (page 2-135) for a timer that changes the timer output to TRUE when timing starts and then changes the timer output to FALSE when the set time is reached.
- Use the TOF instruction (page 2-132) for a timer that starts when *In* changes to FALSE and then changes the timer output to FALSE when the elapsed time reaches the set time.
- To reduce timer execution time, use the Timer instruction (page 2-141), which times in increments of 100 ms.
- If you are connected to an HMI that does not support TIME data, you must convert the set time from integer data to TIME data before you input it to this instruction. Use the NanoSecToTime instruction (page 2-640) to convert integer data to TIME data. Use the TimeToNanoSec instruction (page 2-638) to convert TIME data to integer data. Both instructions express the time in nanoseconds. The user programming for when the INT variable *msIntVar* is the set time in milliseconds is given below.



ST

```
tmpLintVar:=msIntVar*LINT#1000000;
msTimeVar:=NanoSecToTime(tmpLintVar);
TON_instance(In:=Trigger, PT:=msTimeVar, Q=>Tout);
```

Precautions for Correct Use

- The timing error for which *Q* is TRUE for *PT* is –100 ns to (100 ns + 1 task period).
  The above range includes the following:
  - The ±100 ns is the timing error of *ET*.
  - Time *ET* is judged to see if it has reached *PT* every task period. If time *ET* reaches *PT* immediately after the judgement is completed, there is a delay of one task period.
- The time is displayed in increments of 0.001 ms on the Sysmac Studio, but the timing accuracy is 1 ns.
- The timer starts as soon as operation starts if *In* is already TRUE.
- If T#0ms or a negative number is set for *PT*, *Q* will change to TRUE as soon as the value of *In* changes to TRUE.
- You can change the value of *PT* while the value of *In* is TRUE. Operation is as follows:

Timer status	Value of *Q*	Value of *PT* after it is changed	Operation
After completion of timing	TRUE	---	The value of *Q* remains TRUE. The value of *ET* also does not change. (It remains at the value of *PT* before it was changed.)
Timing in progress	FALSE	*PT* ≥ *ET*	Timing is continued. When the value of *ET* reaches the value of *PT*, the value of *Q* changes to TRUE and *ET* is no longer incremented.
		PT < *ET*	The value of *Q* changes to TRUE immediately. Incrementing *ET* stops immediately.

- If this instruction is in a master control region and the master control region is reset, the timer is reset. The value of *ET* changes to 0 and the value of *Q* changes to FALSE.
- If this instruction is not executed due to the execution of a jump instruction (e.g., the JMP instruction), the value of *ET* is not updated. However, timing still continues. Therefore, *ET* is updated to the correct value the next time the instruction is executed.
- If this instruction is used in a ladder diagram, the value of *Q* changes to FALSE if an error occurs in the previous instruction on the rung.

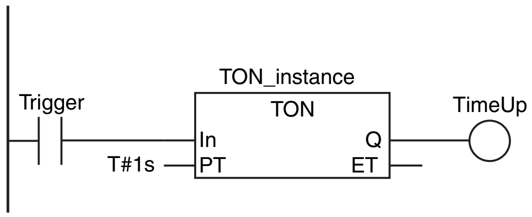
Sample Programming

● Measuring Time with One On-Delay Timer

The value of *TimeUp* will change to TRUE 1 second after the value of *Trigger* changes to TRUE.

LD

Variable	Data type	Initial value	Comment
Trigger	BOOL	FALSE	Execution condition
TimeUp	BOOL	FALSE	Timer output
TON_instance	TON		



ST

Variable	Data type	Initial value	Comment
Trigger	BOOL	FALSE	Execution condition
TimeUp	BOOL	FALSE	Timer output
TON_instance	TON		

```
IF (Trigger=TRUE) THEN
    TON_instance(In:=TRUE, PT:=T#1s, Q=>TimeUp);
ELSE
    TON_instance(In:=FALSE, Q=>TimeUp);
END_IF;
```

The following ST programming performs the same operation.

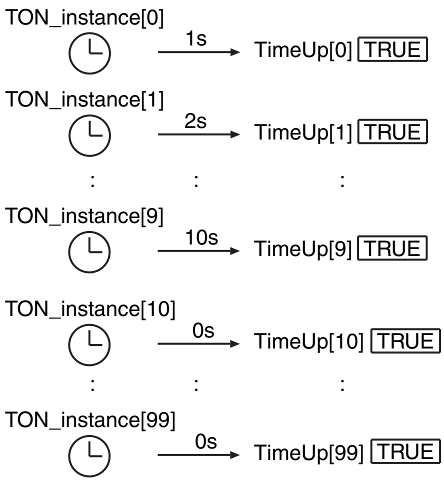
ST

Variable	Data type	Initial value	Comment
Trigger	BOOL	FALSE	Execution condition
TimeUp	BOOL	FALSE	Timer output
TON_instance	TON		

```
TON_instance(In:=Trigger, PT:=T#1s, Q=>TimeUp);
```

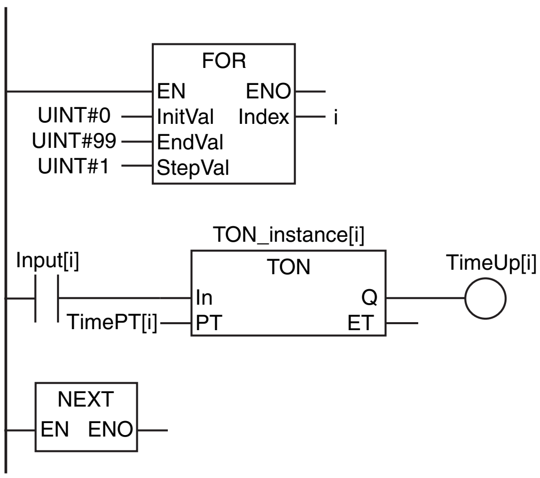
● Measuring Time with Multiple On-Delay Timers

In this example, a total of 100 instances of the On-Delay Timer instruction, TON_instance[0] to TON_instance[99], are programmed. Each timer starts when the value of the corresponding timer input *Input[0]* to *Input[99]* changes to TRUE.
The timers for the first 10 instances, TON_instance[0] to TON_instance[9], change the corresponding values in *TimeUp[i]* to TRUE i+1 seconds (i = 0 to 9) after execution is started.
The timers for the remaining 90 instances, TON_instance[10] to TON_instance[99], change the corresponding values in *TimeUp[i]* (i = 10 to 99) to TRUE as soon as execution is started.



LD

Variable	Data type	Initial value	Comment
Input	ARRAY[0..99] OF BOOL	[100(FALSE)]	Timer input
TimeUp	ARRAY[0..99] OF BOOL	[100(FALSE)]	Timer output
TimePT	ARRAY[0..99] OF TIME	[T#1s, T#2s, T#3s, T#4s, T#5s, T#6s, T#7s, T#8s, T#9s, T#10s, 90(T#0s)]	Set time
TON_instance	ARRAY[0..99] OF TON		
i	UINT	0	Index



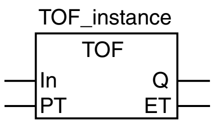
ST

Variable	Data type	Initial value	Comment
Input	ARRAY[0..99] OF BOOL	[100(FALSE)]	Timer input
TimeUp	ARRAY[0..99] OF BOOL	[100(FALSE)]	Timer output
TimePT	ARRAY[0..99] OF TIME	[T#1s, T#2s, T#3s, T#4s, T#5s, T#6s, T#7s, T#8s, T#9s, T#10s, 90(T#0s)]	Set time
TON_instance	ARRAY[0..99] OF TON		
i	UINT	0	Index

```
FOR i  :=UINT#0 TO UINT#99 DO
  TON_instance[i](
    In := Input[i],
    PT := TimePT[i],
    Q   =>TimeUp[i]);
END_FOR;
```

TOF

The TOF instruction outputs FALSE when the set time elapses after the timer starts.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TOF	Off-Delay Timer	FB		TOF_instance (In, PT, Q, ET);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Timer input	Input	TRUE: Timer reset signal FALSE: Timer start signal	Depends on data type.	---	FALSE
PT	Set time		Time from when timer starts until *Q* changes to FALSE	*	ms	0
Q	Timer output	Output	TRUE: Timer output ON FALSE: Timer output OFF	Depends on data type.	---	---
ET	Elapsed time		Elapsed time since timer started	*	ms	

* T#0ms to T#106751d_23h_47m_16s_854.775807ms

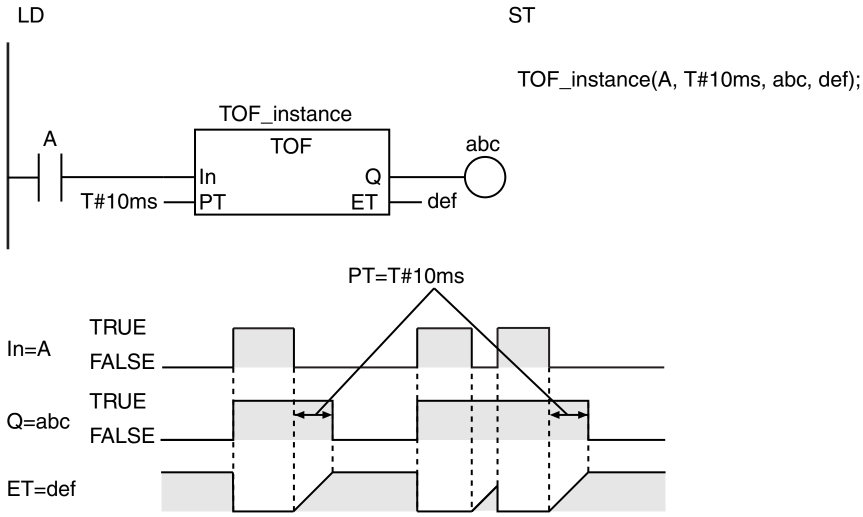
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	OK																			
PT																OK				
Q	OK																			
ET																OK				

Function

The TOF instruction outputs FALSE when the set time elapses after the timer starts. The time is set in nanoseconds. The timer starts when timer input *In* changes to FALSE. Elapsed time *ET* is incremented as time elapses. When *ET* reaches set time *PT*, timer output *Q* changes to FALSE. *ET* is not incremented after that. The timer is reset when *In* changes to TRUE. *ET* changes to 0 and *Q* changes to TRUE.

If the timer is started and then *In* changes to FALSE before *ET* reaches *PT*, the timer is reset.

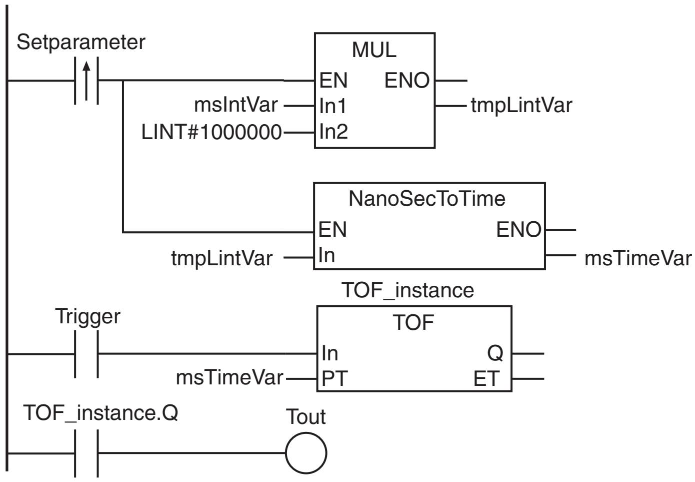
The following figure shows a programming example and timing chart for a *PT* of T#10ms. Variable *abc* will change to FALSE 10 ms after variable *A* changes to FALSE.



Additional Information

- Use the TP instruction (page 2-135) for a timer that changes the timer output to TRUE when timing starts and then changes the timer output to FALSE when the set time is reached.
- Use the TON instruction (page 2-126) for a timer that starts when *In* changes to TRUE and then changes the timer output to TRUE when the elapsed time reaches the set time.
- If you are connected to an HMI that does not support TIME data, you must convert the set time from integer data to TIME data before you input it to this instruction. Use the NanoSecToTime instruction (page 2-640) to convert integer data to TIME data. Use the TimeToNanoSec instruction (page 2-638) to convert TIME data to integer data. Both instructions express the time in nanoseconds. The user programming for when the INT variable *msIntVar* is the set time in milliseconds is given below.

LD



ST

```
tmpLintVar:=msIntVar*LINT#1000000;
msTimeVar:=NanoSecToTime(tmpLintVar);
TOF_instance(In:=Trigger, PT:=msTimeVar, Q=>Tout);
```

Precautions for Correct Use

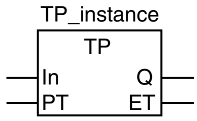
- The timing error for which *Q* is TRUE for *PT* is –100 ns to (100 ns + 1 task period).
  The above range includes the following:
  - The ±100 ns is the timing error of *ET*.
  - Time *ET* is judged to see if it has reached *PT* every task period. If time *ET* reaches *PT* immediately after the judgement is completed, there is a delay of one task period.
- The time is displayed in increments of 0.001 ms on the Sysmac Studio, but the timing accuracy is 1 ns.
- If T#0ms or a negative number is set for *PT*, *Q* will change to FALSE as soon as the value of *In* changes to FALSE.
- The value of *Q* changes to TRUE immediately after execution of this instruction regardless of the value of *In*. *Q* is FALSE from only when the timer is started until the time that is set with *PT* elapses.
- You can change the value of *PT* while the value of *In* is FALSE. Operation is as follows:

Timer status	Value of *Q*	Value of *PT* after it is changed	Operation
After completion of timing	FALSE	---	The value of *Q* remains FALSE. The value of *ET* also does not change. (It remains at the value of *PT* before it was changed.)
Timing in progress	TRUE	*PT* ≥ *ET*	Timing is continued. When the value of *ET* reaches the value of *PT*, the value of *Q* changes to FALSE and *ET* is no longer incremented.
		PT < *ET*	The value of *Q* changes to FALSE immediately. Incrementing *ET* stops immediately.

- If this instruction is in a master control region and the master control region is reset, the operation is as follows:
  - The value of *ET* changes to 0 and the value of *Q* changes to TRUE.
  - If an Out instruction is connected to *Q*, the execution condition to the Out instruction is FALSE.
  - Timing starts as soon as the reset is released.
- If this instruction is not executed due to the execution of a jump instruction (e.g., the JMP instruction), the value of *ET* is not updated. However, timing still continues. Therefore, *ET* is updated to the correct value the next time the instruction is executed.
- If this instruction is used in a ladder diagram, the value of *Q* changes to FALSE if an error occurs in the previous instruction on the rung.

TP

The TP instruction outputs TRUE while the set time elapses after the timer starts.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TP	Timer Pulse	FB		TP_instance (In, PT, Q, ET);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Timer input	Input	TRUE: Timer start signal FALSE: Timer reset signal	Depends on data type.	---	FALSE
PT	Set time		Time that Q remains at TRUE	*	ms	0
Q	Timer output	Output	TRUE: Timer output ON FALSE: Timer output OFF	Depends on data type.	---	---
ET	Elapsed time		Elapsed time since timer started	*	ms	

* T#0ms to T#106751d_23h_47m_16s_854.775807ms

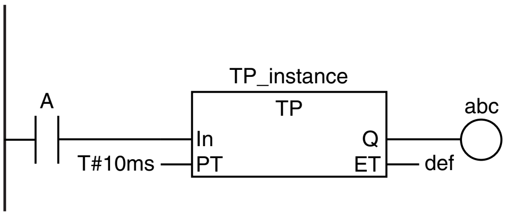
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	OK																			
PT																OK				
Q	OK																			
ET																OK				

Function

The TP instruction outputs TRUE while the set time elapses after the timer starts. The time is set in nanoseconds. The timer starts when timer input *In* changes to TRUE and timer output *Q* changes to TRUE. Elapsed time *ET* is incremented as time elapses. When *ET* reaches set time *PT*, timer output *Q* changes to FALSE. *ET* is not incremented after that. The timer is reset when *In* changes to FALSE. *ET* changes to 0. The timer is not reset even if *In* changes to FALSE after the timer starts but before *ET* reaches *PT*.

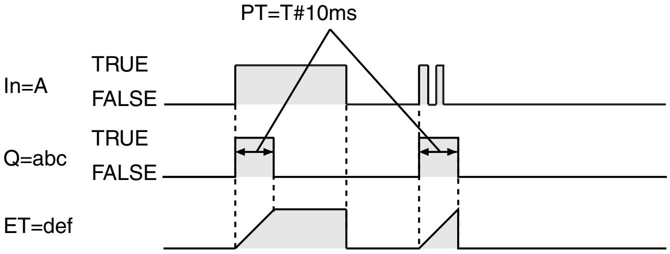
The following figure shows a programming example and timing chart for a *PT* of T#10ms. Variable *abc* changes to TRUE as soon as variable *A* changes to TRUE. Variable *abc* changes to FALSE 10 ms later.

LD



ST

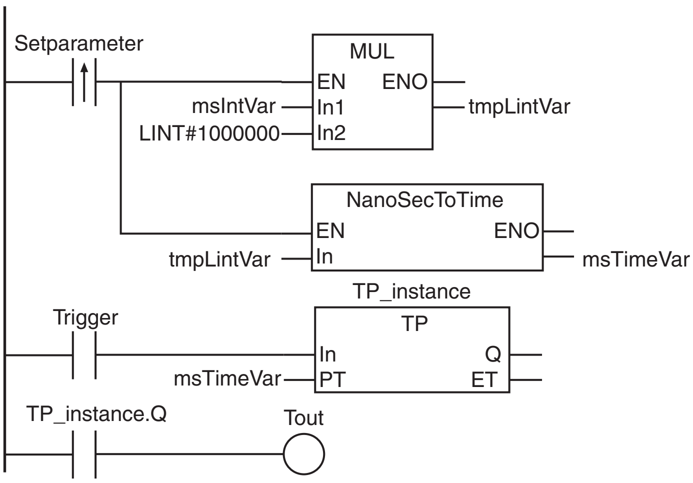
```
TP_instance(A, T#10ms, abc, def);
```



Additional Information

- Use the TON instruction (page 2-126) for a timer that starts when *In* changes to TRUE and then changes the timer output to TRUE when the elapsed time reaches the set time.
- Use the TOF instruction (page 2-132) for a timer that starts when *In* changes to FALSE and then changes the timer output to FALSE when the elapsed time reaches the set time.
- If you are connected to an HMI that does not support TIME data, you must convert the set time from integer data to TIME data before you input it to this instruction. Use the NanoSecToTime instruction (page 2-640) to convert integer data to TIME data. Use the TimeToNanoSec instruction (page 2-638) to convert TIME data to integer data. Both instructions express the time in nanoseconds. The user programming for when the INT variable *msIntVar* is the set time in milliseconds is given below.

LD



ST

```
tmpLintVar:=msIntVar*LINT#1000000;
msTimeVar:=NanoSecToTime(tmpLintVar);
TP_instance(In:=Trigger, PT:=msTimeVar, Q=>Tout);
```

Precautions for Correct Use

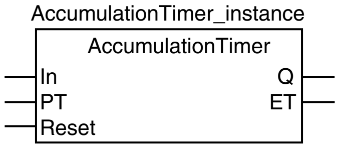
- The timing error for which *Q* is TRUE for *PT* is –100 ns to (100 ns + 1 task period).
  The above range includes the following:
  - The ±100 ns is the timing error of *ET*.
  - Time *ET* is judged to see if it has reached *PT* every task period. If time *ET* reaches *PT* immediately after the judgement is completed, there is a delay of one task period.
- The time is displayed in increments of 0.001 ms on the Sysmac Studio, but the timing accuracy is 1 ns.
- The timer starts as soon as operation starts if *In* is already TRUE.
- If T#0ms or a negative number is set for *PT*, *Q* will not change to TRUE even if the value of *In* changes to TRUE.
- You can change the value of *PT* while the value of *In* is TRUE. Operation is as follows:

Timer status	Value of *Q*	Value of *PT* after it is changed	Operation
After completion of timing	FALSE	---	The value of *Q* remains FALSE. The value of *ET* also does not change. (It remains at the value of *PT* before it was changed.)
Timing in progress	TRUE	*PT* ≥ *ET*	Timing is continued. When the value of *ET* reaches the value of *PT*, the value of Q changes to FALSE and *ET* is no longer incremented.
		PT < *ET*	The value of *Q* changes to FALSE immediately. Incrementing *ET* stops immediately.

- If this instruction is in a master control region and the master control region is reset, timing is continued to the end if the timer is operating. Then, the value of *ET* changes to 0 and the value of *Q* changes to FALSE. However, if an Out instruction is connected to *Q*, the execution condition to the Out instruction is FALSE even if the value of *Q* is TRUE.
- If this instruction is not executed due to the execution of a jump instruction (e.g., the JMP instruction), the value of *ET* is not updated and timing is not performed. Timing restarts when the instruction is executed again.
- If this instruction is used in a ladder diagram, the value of *Q* changes to FALSE if an error occurs in the previous instruction on the rung.

AccumulationTimer

The AccumulationTimer instruction totals the time that the timer input is TRUE.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AccumulationTimer	Accumulation Timer	FB		AccumulationTimer_instance(In, PT, Reset, Q, ET);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Timer input	Input	TRUE: Timer operates FALSE: Timer stops	Depends on data type.	---	FALSE
PT	Set time		Maximum time	*	ms	0
Reset	Reset		TRUE: Timer reset FALSE: Timer not reset	Depends on data type.	---	FALSE
Q	Timer output	Output	TRUE: *ET* reached *PT.* FALSE: *ET has not* reached *PT.*	Depends on data type.	---	---
ET	Total time		Total time	*	ms	

* T#0ms to T#106751d_23h47m_16s_854.775807ms

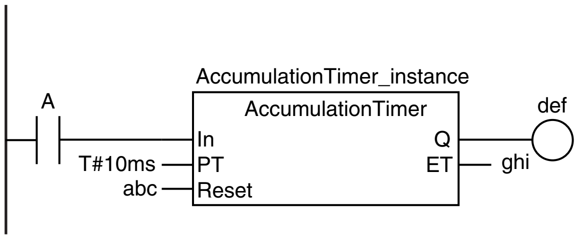
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	OK																			
PT																OK				
Reset	OK																			
Q	OK																			
ET																OK				

Function

The AccumulationTimer instruction totals the time that the timer input is TRUE. The time is set in nanoseconds. If reset *Reset* is FALSE, the timer starts when *In* changes to TRUE. Total time *ET* is incremented as time elapses. The timer stops when *In* changes to FALSE. *ET* is held. When *In* changes to TRUE again, the timer starts again. *ET* is incremented from the value that was previously held. When *ET* reaches set time *PT*, timer output *Q* changes to TRUE. *ET* is not incremented after that. The timer is reset when *Reset* changes to TRUE.
ET changes to 0 and *Q* changes to FALSE.

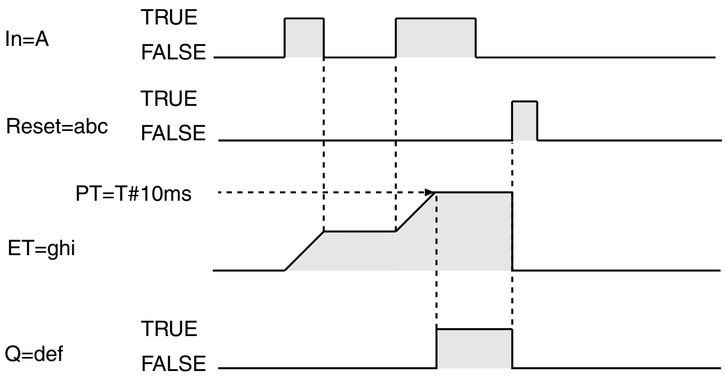
The following figure shows a programming example and timing chart for a *PT* of T#10ms. Variable *abc* changes to TRUE when variable *A* is TRUE for a total of 10 ms (i.e., the total time).

LD



ST

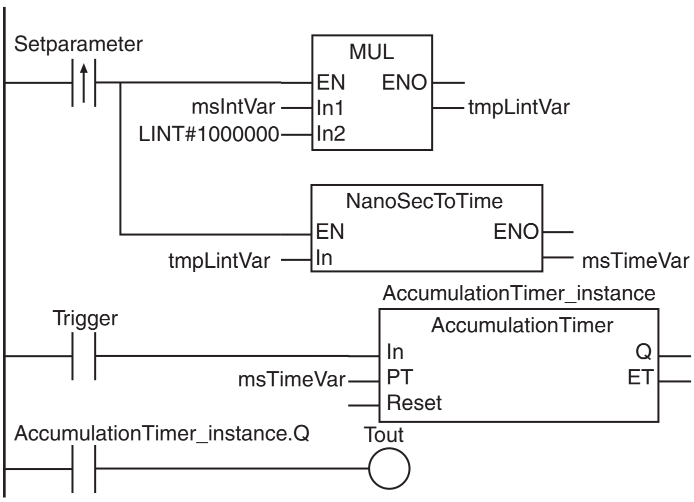
```
AccumulationTimer_instance(A, T#10ms, abc, def, ghi);
```



Additional Information

- Use the TON instruction (page 2-126) for a timer that resets the timer output and elapsed time when *In* changes to FALSE.
- If you are connected to an HMI that does not support TIME data, you must convert the set time from integer data to TIME data before you input it to this instruction. Use the NanoSecToTime instruction (page 2-640) to convert integer data to TIME data. Use the TimeToNanoSec instruction (page 2-638) to convert TIME data to integer data. Both instructions express the time in nanoseconds. The user programming for when the INT variable *msIntVar* is the set time in milliseconds is given below.

LD



ST

```
tmpLintVar:=msIntVar*LINT#1000000;
msTimeVar:=NanoSecToTime(tmpLintVar);
AccumulationTimer_instance(In:=Trigger, PT:=msTimeVar, Q=>Tout);
```

Precautions for Correct Use

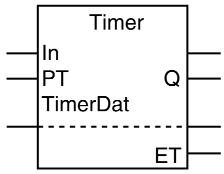
- The timing error for which *Q* is TRUE for *PT* is –100 ns to (100 ns + 1 task period).
  The above range includes the following:
  - The ±100 ns is the timing error of *ET*.
  - Time *ET* is judged to see if it has reached *PT* every task period. If time *ET* reaches *PT* immediately after the judgement is completed, there is a delay of one task period.
- The time is displayed in increments of 0.001 ms on the Sysmac Studio, but the timing accuracy is 1 ns.
- If *In* and *Reset* are both TRUE, *Reset* has priority. That is, *ET* changes to 0 and *Q* changes to FALSE.
- The timer starts as soon as operation starts if *In* is already TRUE.
- If T#0ms or a negative number is set for *PT*, *Q* will change to TRUE as soon as the value of *In* changes to TRUE.
- You can change the value of *PT* before the value of *ET* reaches the value of *PT*. Operation is as follows:

Timer status	Value of *Q*	Value of *PT* after it is changed	Operation
After completion of timing	TRUE	---	The value of *Q* remains TRUE. The value of *ET* also does not change. (It remains at the value of *PT* before it was changed.)
Timing in progress	FALSE	*PT* ≥ *ET*	When the value of *In* changes to TRUE, timing is continued. When the value of *ET* reaches the value of *PT*, the value of *Q* changes to TRUE and *ET* is no longer incremented.
		PT < *ET*	When the value of *In* changes to TRUE, the value of *Q* changes to TRUE immediately. Incrementing *ET* stops immediately.

- If this instruction is in a master control region and the master control region is reset, the operation is as follows:
  - The timer stops. The values of *ET* and *Q* at that time are retained.
  - When the master control reset is cleared, *ET* is incremented again from the value that was retained.
  - If an Out instruction is connected to *Q*, the execution condition to the Out instruction is FALSE even if the value of *Q* is TRUE.
  - *Reset* is enabled.
- If this instruction is not executed due to the execution of a jump instruction (e.g., the JMP instruction), the value of *ET* is not updated. However, timing still continues. Therefore, *ET* is updated to the correct value the next time the instruction is executed.
- If this instruction is used in a ladder diagram, the value of *Q* changes to FALSE if an error occurs in the previous instruction on the rung.

Timer

The Timer instruction outputs TRUE when the set time elapses after the timer starts. The time is set in increments of 100 ms.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Timer	Hundred-ms Timer	FUN		Out:=Timer (In, PT, TimerDat, Q, ET);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Timer input	Input	TRUE: Timer start specification FALSE: Timer reset specification	Depends on data type.	---	FALSE
PT	Set time		Time from when timer starts until *Q* changes to TRUE		ms	*
TimerDat	Timer status	In-out	Current status of timer	---	---	---
Out	Return value	Output	TRUE: Make timer output TRUE FALSE: Make timer output FALSE	Depends on data type.	---	---
Q	Timer output		Same meaning as *Out*.			
ET	Remaining time		Remaining time		ms	

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	OK																			
PT							OK													
TimerDat	Structure _sTimer																			
Out	OK																			
Q	OK																			
ET							OK													

Function

The Timer instruction outputs TRUE when the set time elapses after the timer starts. The time is set in increments of 100 ms.

The timer is reset when timer input *In* changes to FALSE. Remaining time *ET* is set to set time *PT*, and timer output *Q* changes to FALSE.

The timer starts when *In* changes to TRUE. The value of *ET* is timed down. When the value of *ET* reaches 0, timer output *Q* changes to TRUE. *ET* is not timed down after that.

The timer is reset if *In* changes to FALSE after the timer starts but before *ET* reaches 0.

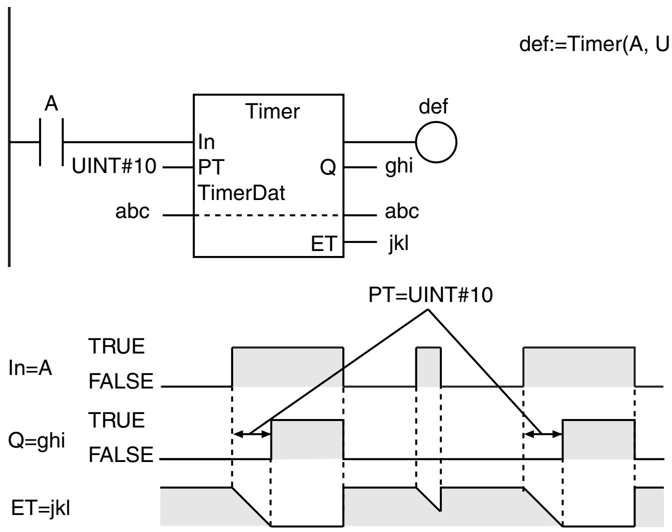
The data type of timer status *TimerDat* is structure _sTimer.

The following figure shows a programming example and timing chart when *PT* is UINT#10. Variable *ghi* will change to TRUE 1,000 ms (1 s) after variable *A* changes to TRUE.

LD

ST

```
def:=Timer(A, UINT#10, abc, ghi, jkl);
```



Additional Information

For more precise timing, use the TON instruction (page 2-126), which is set in nanoseconds. The TON instruction times in nanoseconds when the instruction is executed, so it is more precise than the Timer instruction. However, the execution time of the Timer instruction is shorter.

Precautions for Correct Use

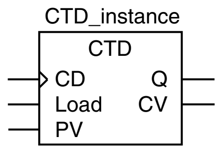
- Timing is performed at the beginning of the POU that contains this instruction. Therefore, the value of *ET* will be the same regardless of where the instruction is executed in the POU.
- The timing error for which *Q* is TRUE for *PT* is + 1 task period.
  The above range includes the following:
  - Time *ET* is judged to see if it has reached *PT* every task period. If time *ET* reaches *PT* immediately after the judgement is completed, there is a delay of one task period.
- Although *TimerDat* is an in-out variable, it is not necessary to pass any values. Create a memory area for the size of the _sTimer structure and pass it to the instruction.
- Do not change the contents of *TimerDat*.
- The timer starts as soon as operation starts if *In* is already TRUE.
- If the value of *PT* changes, the new value is used from the next time that the timer is reset. The value is not updated while timing is in progress.
- If this instruction is in a master control region and the master control region is reset, the timer is reset. *ET* is set to the value of *PT* and the value of *Q* changes to FALSE.

- If this instruction is not executed due to the execution of a jump instruction (e.g., the JMP instruction), the value of *ET* is not updated. However, timing still continues. Therefore, *ET* is updated to the correct value the next time the instruction is executed.
- If this instruction is used in a ladder diagram, the values of *Q* and *Out* change to FALSE if an error occurs in the previous instruction on the rung.

Counter Instructions

CTD

The CTD instruction decrements the counter value when the counter input signal is received. The preset value and counter value must have an INT data type.

Instruction	Name	FB/FUN	Graphic expression	ST expression
CTD	Down-counter	FB		CTD_instance (CD, Load, PV, Q, CV);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
CD	Counter input	Input	Counter input	Depends on data type.	---	FALSE
Load*	Load signal		TRUE: Set CV to PV.			
PV	Preset value		Counter preset value	0 to 32767		0
Q	Counter output	Output	TRUE: Counter output ON FALSE: Counter output OFF	Depends on data type.	---	---
CV	Counter value		Counter present value	0 to 32767		

* On Sysmac Studio version 1.03, you can use "LD" instead of "Load" to more clearly show the correspondence between the variables and the parameter names in ST expressions. For example, you can use the following notation: *CTD_instance* (*CD:=A*, *LD:=abc*, *PV:=INT#5*, *Q=>def*, *CV=>ghi*);.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
CD	OK																			
Load	OK																			
PV											OK									
Q	OK																			
CV											OK									

Function

The CTD instruction creates a down counter. The preset value and counter value must have an INT data type.

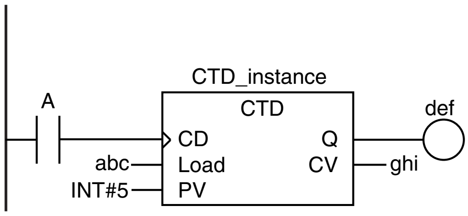
When load signal *Load* changes to TRUE, counter value CV is set to the value of preset value *PV* and counter output *Q* changes to FALSE. When counter input signal *CD* changes to TRUE, *CV* is decremented. When the value of *CV* reaches 0 or less, the value of *Q* changes to TRUE.

After the value of *CV* reaches 0 or less, *CV* does not change even if *CD* changes to TRUE.

CD is ignored while *Load* is TRUE. *CV* is not decremented.

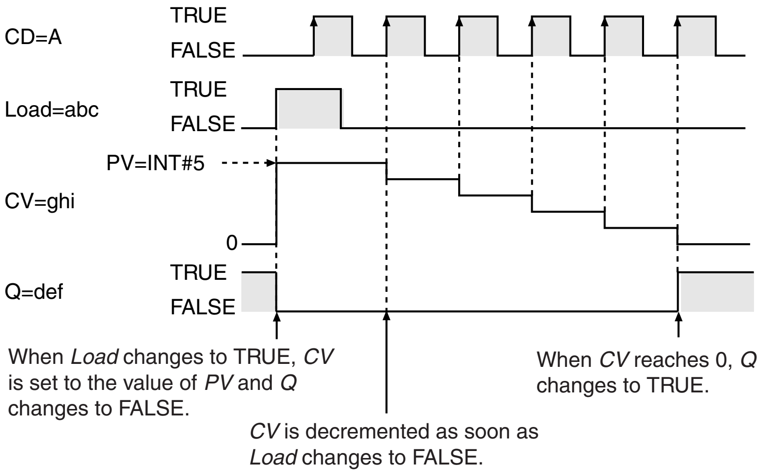
The following figure shows a programming example and timing chart for a *PV* of INT#5.

LD



ST

```
CTD_instance(A, abc, INT#5, def, ghi);
```



Additional Information

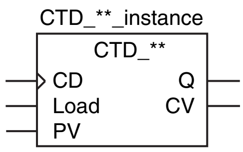
- Use the CTU instruction (page 2-150) to create a counter that increments the counter value each time the counter input signal is received.
- Use the CTUD instruction (page 2-155) to create a counter that is both incremented and decremented.

Precautions for Correct Use

- Change *Load* to TRUE and then back to FALSE to restart a counter that has completed counting down.
- Even when *PV* is set to a negative value, *CV* is set to the value of *PV* when the value of *Load* changes to TRUE. The value of *CV* will be 0 or less, so the value of *Q* changes to TRUE immediately. After that, the value of *CV* is not decremented even if the value of *CD* changes.
- If the value of *CD* is FALSE and the power supply is interrupted or the operating mode is changed to PROGRAM mode, the value of *CV* is decremented once if the value of *CD* is TRUE when instruction execution is restarted.
- If this instruction is used in a ladder diagram, the value of *Q* changes to FALSE if an error occurs in the previous instruction on the rung.

CTD_**

The CTD_** instruction decrements the counter value when the counter input signal is received. The preset value and counter value must be one of the following data types: DINT, LINT, UDINT, or ULINT.

Instruction	Name	FB/FUN	Graphic expression	ST expression
CTD_**	Down-counter Group	FB	 "**" must be DINT, LINT, UDINT, or ULINT.	CTD_**_instance (CD, Load, PV, Q, CV); "**" must be DINT, LINT, UDINT, or ULINT.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
CD	Counter input	Input	Counter input	Depends on data type.	---	FALSE
Load[*1]	Load signal		TRUE: Set CV to PV.			
PV	Preset value		Counter preset value	Depends on data type.[*2]		0
Q	Counter output	Output	TRUE: Counter output ON FALSE: Counter output OFF	Depends on data type.	---	---
CV	Counter value		Counter present value	Depends on data type.[*2]		

*1 On Sysmac Studio version 1.03, you can use “LD” instead of “Load” to more clearly show the correspondence between the variables and the parameter names in ST expressions. For example, you can use the following notation:*CTD_LINT_instance*(*CD:=A, LD:=abc, PV:=LINT#5, Q=>def, CV=>ghi*);.

*2 Negative numbers are excluded.

	Boolean	Bit string				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
CD	OK																			
Load	OK																			
PV								OK	OK			OK	OK							
Q	OK																			
CV	Must be the same data type as *PV*																			

Function

A CTD_** instruction creates a down counter. The preset value and counter value must be one of the following data types: DINT, LINT, UDINT, or ULINT. The name of the instruction is determined by the data type of *PV* and *CV*. For example, if they are the CV data type, the instruction is CTD_LINT.

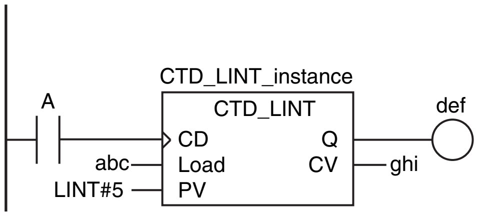
When load signal *Load* changes to TRUE, counter value *CV* is set to the value of preset value *PV* and counter output *Q* changes to FALSE. When counter input signal *CD* changes to TRUE, *CV* is decremented. When the value of *CV* reaches 0 or less, the value of *Q* changes to TRUE.

After the value of *CV* reaches 0 or less, *CV* does not change even if *CD* changes to TRUE.

CD is ignored while *Load* is TRUE. *CV* is not decremented.

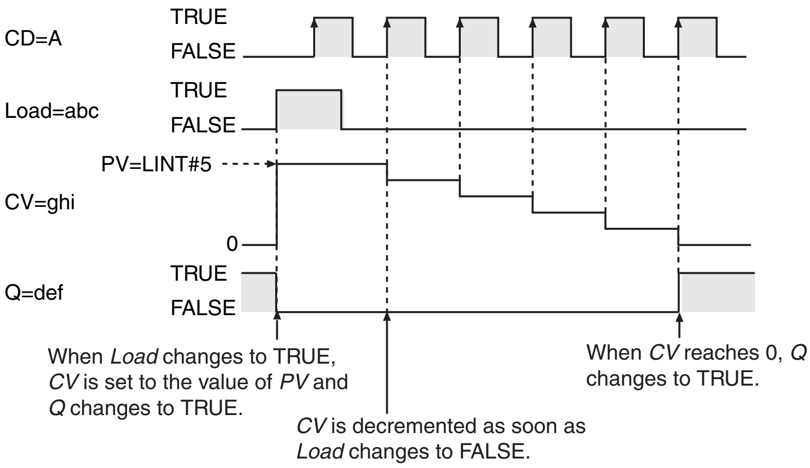
The following figure shows a CTD_LINT programming example and timing chart for a *PV* of LINT#5.

LD



ST

```
CTD_LINT_instance(A, abc, LINT#5, def, ghi);
```



Additional Information

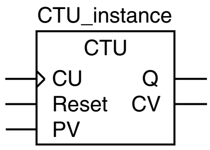
- Use the CTU instruction (page 2-150) to create a counter that increments the counter value each time the counter input signal is received.
- Use the CTUD instruction (page 2-155) to create a counter that is both incremented and decremented.

Precautions for Correct Use

- Change *Load* to TRUE and then back to FALSE to restart a counter that has completed counting down.
- Use the same data type for *PV* and *CV*.
- Even when *PV* is set to a negative value, *CV* is set to the value of *PV* when the value of *Load* changes to TRUE. The value of *CV* will be 0 or less, so the value of Q changes to TRUE immediately. After that, the value of *CV* is not decremented even if the value of *CD* changes.
- If the value of *CD* is FALSE and the power supply is interrupted or the operating mode is changed to PROGRAM mode, the value of *CV* is decremented once if the value of *CD* is TRUE when instruction execution is restarted.
- If this instruction is used in a ladder diagram, the value of *Q* changes to FALSE if an error occurs in the previous instruction on the rung.

CTU

The CTU instruction increments the counter value when the counter input signal is received. The preset value and counter value must have an INT data type.

Instruction	Name	FB/FUN	Graphic expression	ST expression
CTU	Up-counter	FB		CTU_instance (CU, Reset, PV, Q, CV);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
CU	Counter input	Input	Counter input	Depends on data type.	---	FALSE
Reset*	Reset signal		TRUE: Reset *CV* to 0.			
PV	Preset value		Counter preset value	0 to 32767		0
Q	Counter output	Output	TRUE: Counter output ON FALSE: Counter output OFF	Depends on data type.	---	---
CV	Counter value		Counter present value	0 to 32767		

* On Sysmac Studio version 1.03, you can use “R” instead of “Reset” to more clearly show the correspondence between the variables and the parameter names in ST expressions. For example, you can use the following notation: *CTU_instance* (*CU:=A, R:=abc, PV:=INT#5, Q=>def, CV=>ghi*);.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
CU	OK																			
Reset	OK																			
PV											OK									
Q	OK																			
CV											OK									

Function

The CTU instruction creates an up counter. The preset value and counter value must have an INT data type.

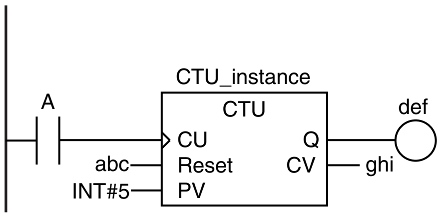
When reset signal *Reset* changes to TRUE, counter value *CV* changes to 0 and counter output *Q* changes to FALSE. When counter input signal *CU* changes to TRUE, *CV* is incremented. When the value of *CV* reaches the value of *PV* or higher, the value of *Q* changes to TRUE.

After the value of *CV* reaches the value of *PV* or higher, the value of *CV* does not change even if the value of *CU* changes to TRUE.

CU is ignored while *Reset* is TRUE. *CV* is not incremented.

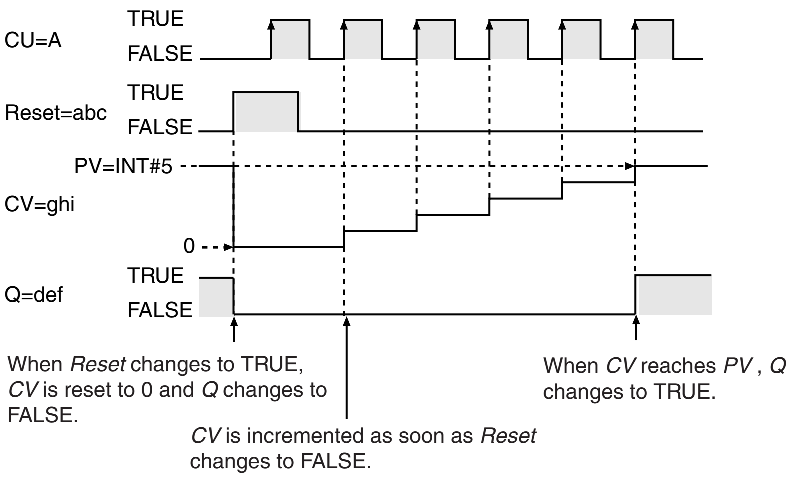
The following figure shows a programming example and timing chart for a *PV* of INT#5.

LD



ST

CTU_instance(A, abc, INT#5, def, ghi);



Additional Information

- Use the CTD instruction (page 2-146) to create a counter that decrements the counter value each time the counter input signal is received.
- Use the CTUD instruction (page 2-155) to create a counter that is both incremented and decremented.

Precautions for Correct Use

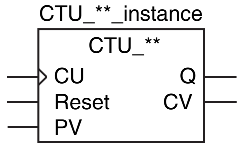
- Change *Reset* to TRUE and then back to FALSE to restart a counter that has completed counting up.
- Even when *PV* is set to a negative value, *CV* is set to 0 when the value of *Reset* changes to TRUE. The value of *CV* will be higher than the value of *PV*, so the value of *Q* changes to TRUE immediately. After that, the value of *CV* is not incremented even if the value of *CU* changes.
- The following operation is performed if the value of *PV* changes while the value of *Reset* is FALSE.

Value of *PV*	Meaning
Larger than the current value of *CV*	The count operation is continued.
Equal to or smaller than the current value of *CV*	The count operation is ended. The value of *Q* changes to TRUE. The current value of *CV* is retained. It does not change.

- If the value of *CU* is FALSE and the power supply is interrupted or the operating mode is changed to PROGRAM mode, the value of *CV* is incremented once if the value of *CU* is TRUE when instruction execution is restarted.
- If this instruction is used in a ladder diagram, the value of *Q* changes to FALSE if an error occurs in the previous instruction on the rung.

CTU_**

The CTU_** instruction increments the counter value when the counter input signal is received. The preset value and counter value must be one of the following data types: DINT, LINT, UDINT, or ULINT.

Instruction	Name	FB/FUN	Graphic expression	ST expression
CTU_**	Up-counter Group	FB	 "**" must be DINT, LINT, UDINT, or ULINT.	CTU_**_instance (CU, Reset, PV, Q, CV); "**" must be DINT, LINT, UDINT, or ULINT.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
CU	Counter input	Input	Counter input	Depends on data type.	---	FALSE
Reset*1	Reset signal		TRUE: Reset *CV* to 0.			
PV	Preset value		Counter preset value	Depends on data type.*2		0
Q	Counter output	Output	TRUE: Counter output ON FALSE: Counter output OFF	Depends on data type.	---	---
CV	Counter value		Counter present value	Depends on data type.*2		

*1 On Sysmac Studio version 1.03, you can use “R” instead of “Reset” to more clearly show the correspondence between the variables and the parameter names in ST expressions. For example, you can use the following notation: *CTU_LINT_instance*(*CU:=A, R:=abc, PV:=LINT#5, Q=>def, CV=>ghi*);.

*2 Negative numbers are excluded.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
CU	OK																			
Reset	OK																			
PV								OK	OK			OK	OK							
Q	OK																			
CV	Must be the same data type as *PV*																			

Function

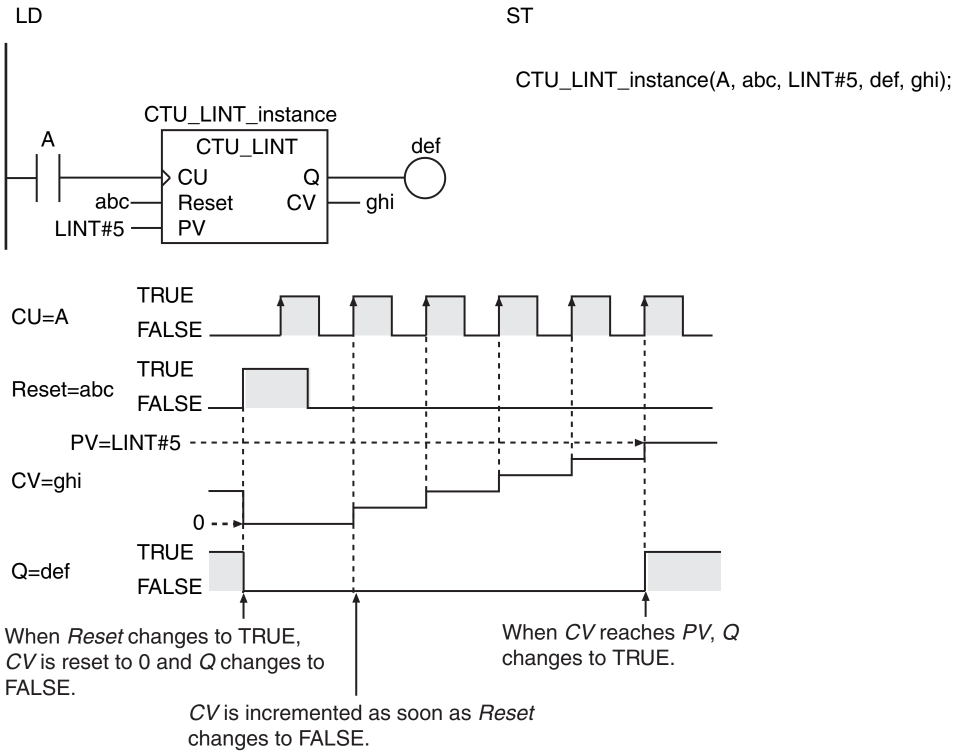
A CTU_** instruction creates an up counter. The preset value and counter value must be one of the following data types: DINT, LINT, UDINT, or ULINT. The name of the instruction is determined by the data type of *PV* and *CV*. For example, if they are the LINT data type, the instruction is CTU_LINT.

When reset signal *Reset* changes to TRUE, counter value *CV* changes to 0 and counter output *Q* changes to FALSE. When counter input signal *CU* changes to TRUE, *CV* is incremented. When the value of *CV* reaches the value of *PV* or higher, the value of *Q* changes to TRUE.

After the value of *CV* reaches the value of *PV* or higher, the value of *CV* does not change even if the value of *CU* changes to TRUE.

CU is ignored while *Reset* is TRUE. *CV* is not incremented.

The following figure shows a CTU_LINT programming example and timing chart for a *PV* of LINT#5.



Additional Information

- Use the CTD instruction (page 2-146) to create a counter that decrements the counter value each time the counter input signal is received.
- Use the CTUD instruction (page 2-155) to create a counter that is both incremented and decremented.

Precautions for Correct Use

- Change *Reset* to TRUE and then back to FALSE to restart a counter that has completed counting up.
- Even when *PV* is set to a negative value, *CV* is set to 0 when the value of *Reset* changes to TRUE. The value of *CV* will be higher than the value of *PV*, so the value of *Q* changes to TRUE immediately. After that, the value of *CV* is not incremented even if the value of *CU* changes.
- Use the same data type for *PV* and *CV*.
- The following operation is performed if the value of *PV* changes while the value of *Reset* is FALSE.

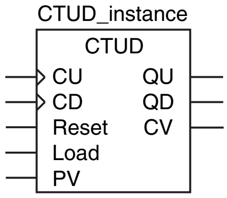
Value of *PV*	Meaning
Larger than the current value of *CV*	The count operation is continued.
Equal to or smaller than the current value of *CV*	The count operation is ended. The value of *Q* changes to TRUE. The current value of *CV* is retained. It does not change.

- If the value of *CU* is FALSE and the power supply is interrupted or the operating mode is changed to PROGRAM mode, the value of *CV* is incremented once if the value of *CU* is TRUE when instruction execution is restarted.

- If this instruction is used in a ladder diagram, the value of *Q* changes to FALSE if an error occurs in the previous instruction on the rung.

CTUD

The CTUD instruction creates an up-down counter that operates according to an up-counter input and a down-counter input. The preset value and counter value must have an INT data type.

Instruction	Name	FB/FUN	Graphic expression	ST expression
CTUD	Up-down Counter	FB		CTUD_instance (CU, CD, Reset, Load, PV, QU, QD, CV);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
CU	Up-counter input	Input	Up counter input	Depends on data type.	---	FALSE
CD	Down-counter input	Input	Down counter input	Depends on data type.	---	FALSE
Reset*	Reset signal	Input	TRUE: Reset *CV* to 0.	Depends on data type.	---	FALSE
Load*	Load signal	Input	TRUE: *CV* set to *PV*.	Depends on data type.	---	FALSE
PV	Preset value	Input	The final counter value when operating as an up counter The initial counter value when operating as a down counter	0 to 32767	---	0
QU	Up-counter output	Output	TRUE: up-counter output ON FALSE: up-counter output OFF	Depends on data type.	---	---
QD	Down-counter output	Output	TRUE: down-counter output ON FALSE: down-counter output OFF	Depends on data type.	---	---
CV	Counter value	Output	Counter present value	0 to 32767	---	---

* On Sysmac Studio version 1.03, you can use “R” instead of “Reset” and “LD” instead of “Load” to more clearly show the correspondence between the variables and the parameter names in ST expressions. For example, you can use the following notation: *CTUD_instance*(*CU:=A, CD:=B, R:=abc, LD:=def, PV:=INT#3, QU=>ghi, QD=>jkl, CV=>mno*);.

	Boolean	Bit string				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
CU	OK																			
CD	OK																			
Reset	OK																			
Load	OK																			
PV											OK									
QU	OK																			
QD	OK																			
CV											OK									

Function

The CTUD instruction creates an up-down counter that operates according to an up-counter input signal and a down-counter input signal. It has the functions of both an up counter and a down counter. The preset value and counter value must have an INT data type.

Operation as an Up Counter

When reset signal *Reset* changes to TRUE, counter value *CV* changes to 0 and up-counter output *QU* changes to FALSE. When up-counter input signal *CU* changes to TRUE, *CV* is incremented. When the value of *CV* reaches the value of *PV* or higher, the value of *QU* changes to TRUE. After the value of *CV* reaches the value of *PV* or higher, the value of *CV* does not change even if the value of *CU* changes to TRUE.

Operation as a Down Counter

When load signal *Load* changes to TRUE, counter value *CV* changes to the value of preset value *PV* and down-counter output *QD* changes to FALSE. When down-counter input signal *CD* changes to TRUE, *CV* is decremented. When the value of *CV* reaches 0 or less, the value of *QD* changes to TRUE. After the value of *CV* reaches 0 or less, *CV* does not change even if *CD* changes to TRUE.

Common Operation for Up and Down Counters

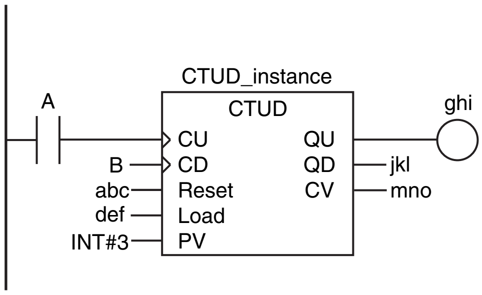
CU and *CD* are ignored while *Load* and *Reset* are TRUE. *CV* is not incremented or decremented. If both *CU* and *CD* change to TRUE at the same time, *CV* will not change. If *Reset* and *Load* are both TRUE, *Reset* has priority and the value of *CV* changes to 0. If *Reset* changes to TRUE, *CV* changes to 0, and so *QD* changes to TRUE. If *Load* changes to TRUE, the value of *CV* changes to *PV*, and so *QU* changes to TRUE.

The following table shows the relationship between *Reset*, *Load*, *CV*, *QU*, and *QD*. This assumes that the value of PV is larger than 0.

Reset	***Load***	***CV***	***QU***	***QD***	**Operation**
FALSE	FALSE	0 or lower	FALSE	TRUE	Only up counter operation is performed. • *CV* is incremented when *CU* changes to TRUE. It is not decremented when *CD* changes to TRUE.
		Between 0 and *PV*	FALSE	FALSE	Both up and down counter operation is performed. • *CV* is incremented when *CU* changes to TRUE and decremented when *CD* changes to TRUE.
		PV or higher	TRUE	FALSE	Only down counter operation is performed. • *CV* is decremented when *CD* changes to TRUE. It is not incremented when *CU* changes to TRUE.
TRUE	FALSE	0	FALSE	TRUE	The up counter is reset. • The value of *CV* is set to 0.
FALSE	TRUE	*PV*	TRUE	FALSE	The down counter is reset. • The value of *CV* is set to *PV*.
TRUE	TRUE	0	FALSE	TRUE	The up counter is reset. *Reset* take priority over *Load*. • The value of *CV* is set to 0.

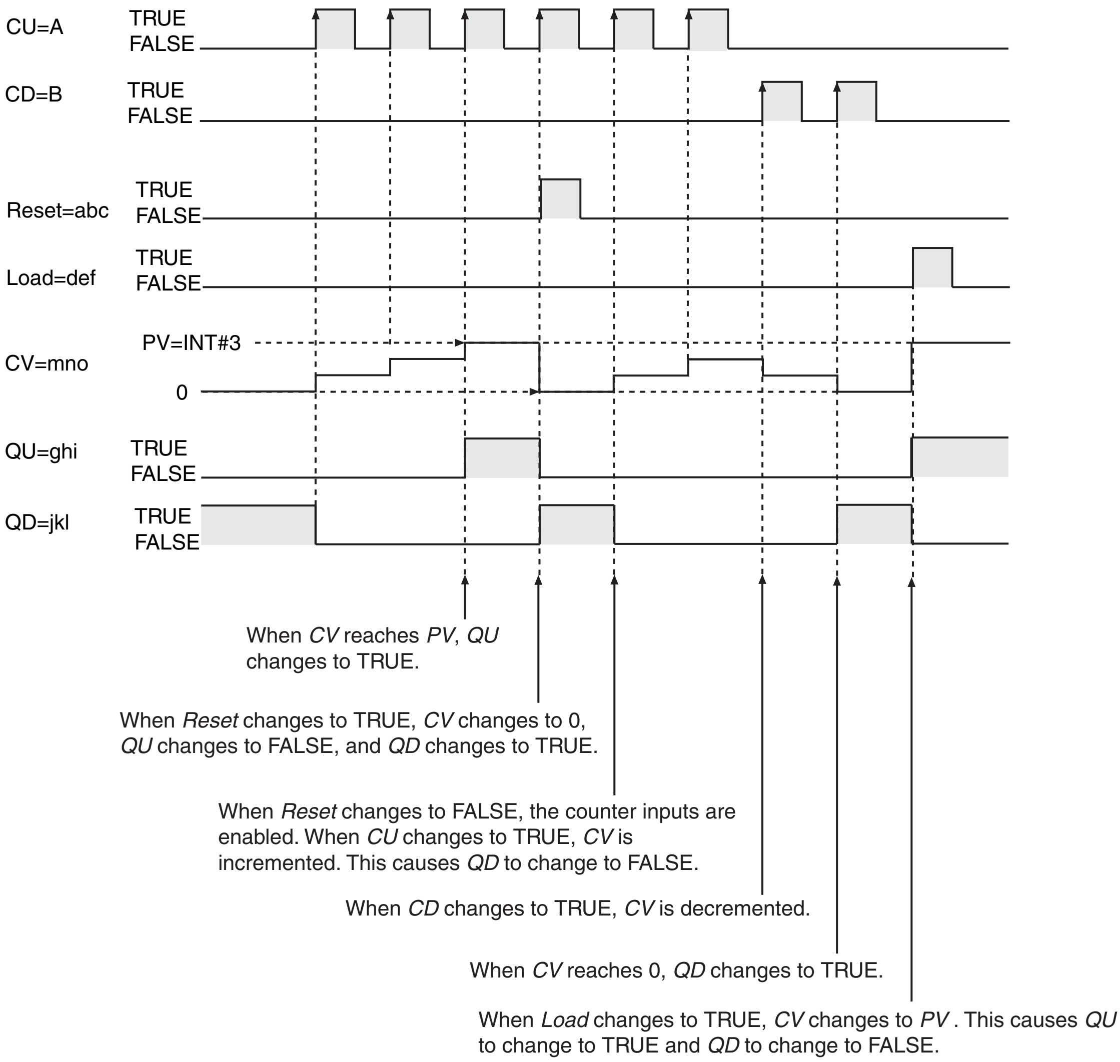
The following figure shows a programming example and timing chart for a *PV* of INT#3.

LD



ST

```
CTUD_instance(A, B, abc, def, INT#3, ghi, jkl, mno);
```



Additional Information

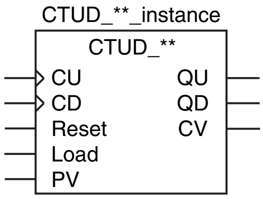
Use the CTD instruction (page 2-146) or CTU instruction (page 2-150) to create a counter that only decrements or only increments.

Precautions for Correct Use

- If you change *Reset* to TRUE to reset the up-counter operation, *QU* will change to FALSE and *QD* will change to TRUE.
- If you change *Load* to TRUE to reset the down-counter operation, *QD* will change to FALSE and *QU* will change to TRUE.
- Even when *PV* is set to a negative value, *CV* is set to the value of *PV* when the value of *Load* changes to TRUE. The value of *CV* will be 0 or less, so the value of *QD* changes to TRUE immediately. After that, the value of *CV* is not decremented even if the value of *CD* changes. When the value of *Reset* changes to TRUE, the value of *CV* changes to 0. The value of *CV* will be the value of *PV* or higher, so the value of *QU* changes to TRUE immediately. After that, the value of *CV* is not incremented even if the value of *CU* changes.
- You can change the value of *PV* during execution of the instruction. If the new value of *PV* is less than the current value of *CV*, the value of *QU* changes to TRUE immediately.
- If the value of *CU* or *CD* is FALSE and the power supply is interrupted or the operating mode is changed to PROGRAM mode, the value of *CV* is incremented or decremented once if the value of *CU* or *CD* is TRUE when instruction execution is restarted.

CTUD_**

The CTUD_** instruction creates an up-down counter that operates according to an up-counter input and a down-counter input. The preset value and counter value must be one of the following data types: DINT, LINT, UDINT, or ULINT.

Instruction	Name	FB/FUN	Graphic expression	ST expression
CTUD_**	Up-down Counter Group	FB	 "**" must be DINT, LINT, UDINT, or ULINT.	CTUD_**_instance (CU, CD, Reset, Load, PV, QU, QD, CV); "**" must be DINT, LINT, UDINT, or ULINT.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
CU	Up-counter input	Input	Up counter input	Depends on data type.	---	FALSE
CD	Down-counter input		Down counter input			
Reset*1	Reset signal		TRUE: Reset *CV* to 0.			
Load*1	Load signal		TRUE: *CV* set to *PV*.			
PV	Preset value		The final counter value when operating as an up counter The initial counter value when operating as a down counter	Depends on data type.*2		0
QU	Up-counter output	Output	TRUE: up-counter output ON FALSE: up-counter output OFF	Depends on data type.	---	---
QD	Down-counter output		TRUE: down-counter output ON FALSE: down-counter output OFF			
CV	Counter value		Counter present value	Depends on data type.*2		

*1 On Sysmac Studio version 1.03, you can use “R” instead of “Reset” and “LD” instead of “Load” to more clearly show the correspondence between the variables and the parameter names in ST expressions. For example, you can use the following notation: *CTUD_LINT_instance*(*CU:=A, CD:=B, R:=abc, LD:=def, PV:=LINT#3, QU=>ghi, QD=>jkl, CV=>mno*);.

*2 Negative numbers are excluded.

	Boolean	Bit string				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
CU	OK																			
CD	OK																			
Reset	OK																			
Load	OK																			
PV								OK	OK			OK	OK							
QU	OK																			
QD	OK																			
CV	Must be the same data type as *PV*																			

Function

A CTUD_** instruction creates an up-down counter that operates according to an up-counter input signal and a down-counter input signal. The counter has the functions of both an up counter and a down counter. The preset value and counter value must be one of the following data types: DINT, LINT, UDINT, or ULINT. The name of the instruction is determined by the data type of *PV* and *CV*. For example, if they are the LINT data type, the instruction is CTUD_LINT.

Operation as an Up Counter

When reset signal *Reset* changes to TRUE, counter value *CV* changes to 0 and up-counter output *QU* changes to FALSE. When up-counter input signal *CU* changes to TRUE, *CV* is incremented. When the value of *CV* reaches the value of *PV* or higher, the value of *QU* changes to TRUE. After the value of *CV* reaches the value of *PV* or higher, the value of *CV* does not change even if the value of *CU* changes to TRUE.

Operation as a Down Counter

When load signal *Load* changes to TRUE, counter value *CV* changes to the value of preset value *PV* and down-counter output *QD* changes to FALSE. When down-counter input signal *CD* changes to TRUE, *CV* is decremented. When the value of *CV* reaches 0 or less, the value of *QD* changes to TRUE. After the value of *CV* reaches 0 or less, *CV* does not change even if *CD* changes to TRUE.

Common Operation for Up and Down Counters

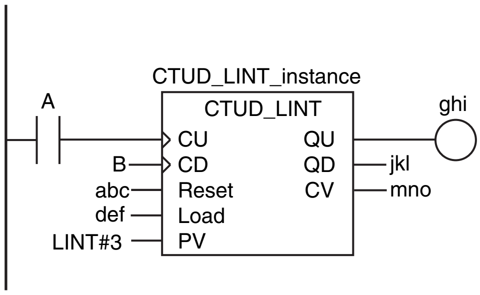
CU and *CD* are ignored while *Load* or *Reset* is TRUE. *CV* is not incremented or decremented. If both *CU* and *CD* change to TRUE at the same time, *CV* will not change. If *Reset* and *Load* are both TRUE, *Reset* has priority and the value of *CV* changes to 0. If *Reset* changes to TRUE, *CV* changes to 0, and so *QD* changes to TRUE. If *Load* changes to TRUE, the value of *CV* changes to *PV*, and so *QU* changes to TRUE.

The following table shows the relationship between *Reset, Load, CV, QU,* and *QD*. This assumes that the value of *PV* is larger than 0.

Reset	***Load***	***CV***	***QU***	***QD***	**Operation**
FALSE	FALSE	0 or lower	FALSE	TRUE	Only up counter operation is performed. • *CV* is incremented when *CU* changes to TRUE. It is not decremented when *CD* changes to TRUE.
		Between 0 and *PV*	FALSE	FALSE	Both up and down counter operation is performed. • *CV* is incremented when *CU* changes to TRUE and decremented when *CD* changes to TRUE.
		PV or higher	TRUE	FALSE	Only down counter operation is performed. • *CV* is decremented when *CD* changes to TRUE. It is not incremented when *CU* changes to TRUE.
TRUE	FALSE	0	FALSE	TRUE	The up counter is reset. • The value of *CV* is set to 0.
FALSE	TRUE	*PV*	TRUE	FALSE	The down counter is reset. • The value of *CV* is set to *PV*.
TRUE	TRUE	0	FALSE	TRUE	The up counter is reset. *Reset* take priority over *Load*. • The value of *CV* is set to 0.

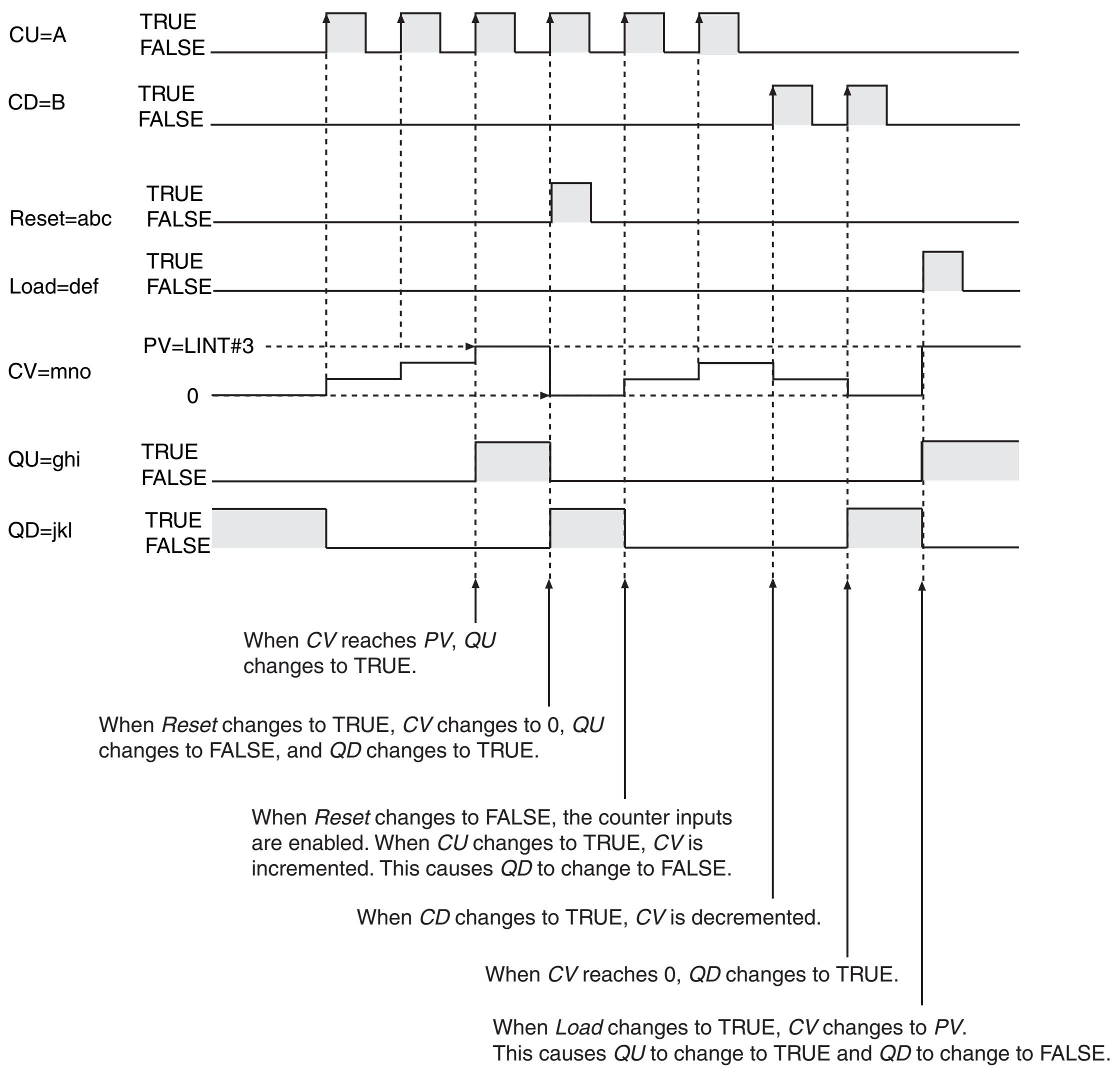
The following figure shows a CTUD_LINT programming example and timing chart for a *PV* of LINT#3.

LD



ST

```
CTUD_LINT_instance(A, B, abc, def, LINT#3, ghi, jkl, mno);
```



Additional Information

Use the CTD instruction (page 2-146) or CTU instruction (page 2-150) to create a counter that only decrements or only increments.

Precautions for Correct Use

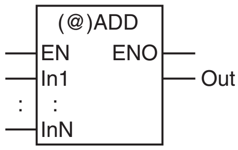
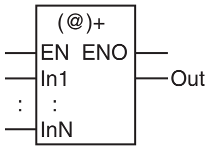
- If you change *Reset* to TRUE to reset the up-counter operation, *QU* will change to FALSE and *QD* will change to TRUE.
- If you change *Load* to TRUE to reset the down-counter operation, *QD* will change to FALSE and *QU* will change to TRUE.
- Even when *PV* is set to a negative value, *CV* is set to the value of *PV* when the value of *Load* changes to TRUE. The value of *CV* will be 0 or less, so the value of *QD* changes to TRUE immediately. After that, the value of *CV* is not decremented even if the value of *CD* changes. When the value of Reset changes to TRUE, the value of *CV* changes to 0. The value of *CV* will be the value of *PV* or higher, so the value of *QU* changes to TRUE immediately. After that, the value of *CV* is not incremented even if the value of *CU* changes.
- You can change the value of *PV* during execution of the instruction. If the new value of *PV* is less than the current value of *CV*, the value of *QU* changes to TRUE immediately.
- Use the same data type for *PV* and *CV*.

- If the value of *CU* or *CD* is FALSE and the power supply is interrupted or the operating mode is changed to PROGRAM mode, the value of *CV* is incremented or decremented once if the value of *CU* or *CD* is TRUE when instruction execution is restarted.

Math Instructions

ADD (+)

The ADD (+) instruction adds integers or real numbers. It also joins text strings.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ADD (+)	Addition	FUN		Out:=In1 + In2;

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1 to InN	Add values	Input	Numbers to add Ladder diagram: N = 2 to 5 ST: N = 2*1	Depends on data type.	---	0*2
Out	Output value	Output	Output value	Depends on data type.	---	---

*1 However, you can use more instructions if you use them as operators in an expression, such as *result*:= *val1* + *val2* + *val3*;. You can use up to 64 instructions in one expression.

*2 If you omit the input parameter that connects to *InN*, the default value is not applied, and a building error will occur. For example, if N is 3 and the input parameters that connect to *In1* and *In2* are omitted, the default values are applied, but if the input parameter that connects to *In3* is omitted, a building error will occur.

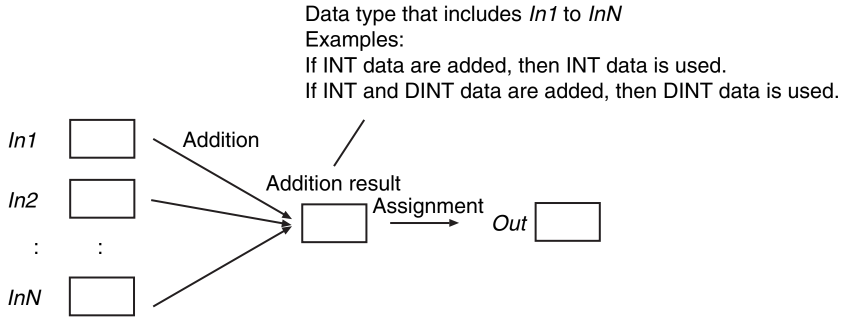
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					OK
Out						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					OK

Function

In a ladder diagram, the Add (+) instruction adds between two and five integers or real numbers and outputs the result to output value *Out*. In ST, the Add instruction adds two integers or real numbers and outputs the result to output value *Out*.

Add values *In1* to *InN* can have different data types. However, use the combination that data types to include are existed. If they are different, calculations are performed with the data type that includes the range of all of the data types. For example, if *In1* is INT data and *In2* is DINT data, calculations are performed with DINT data. Here, the addition result is DINT data.

Refer to *Data Type Ranking Table* and *Casting Rules* in the *NJ/NX-series CPU Unit Software User's Manual* (Cat. No. W501) or *NY-series Industrial Panel PC / Industrial Box PC Software User's Manual* (Cat. No. W558) for the including relationship on data types.



Processing for Overflows

An overflow occurs if the sum of *In1* to *InN* exceeds the valid range of the data type of the addition result. If an overflow occurs, the data types of *In1* to *InN*, the data type of the addition result, and the value of the addition result will be as shown in the following table.

Data types of *In1* to *InN*	Data type of addition result	Value of addition result
All integer data	Integer data	Of the sum of *In1* to *InN*, the addition result will be the value that can be expressed by the number of bits in the data type of the addition result.*1
At least one real number	Real number data	±∞*2

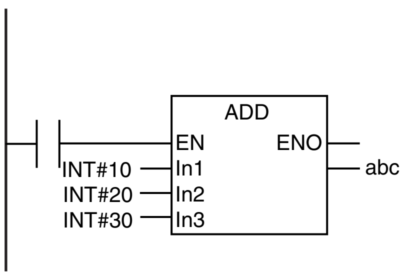
*1 For example, if the value of *In1* is INT#32767 and the value of *In2* is INT#3, the addition result will be INT data. The value of the addition result will be the lower 16 bits of the sum (32,770), i.e., INT#-32766.

*2 If the sum of *In1* to *InN* is positive, the addition result will be positive infinity. If the sum is negative, the addition result will be negative infinity.

Notation Examples

The following example is for when *In1* is INT#10, *In2* is INT#20 and *In3* is INT#30. The value of INT variable *abc* will be INT#60.

LD



ST

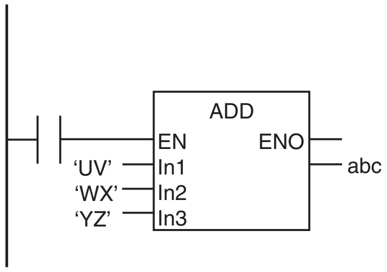
```
abc:=INT#10+INT#20+INT#30;
```

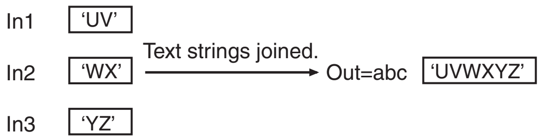
The ADD instruction adds *In1* through *InN*.
The calculation is 10 + 20 + 30 = 60, so the value of *abc* will be INT#60.

Joining Text Strings

If *In1* to *InN* are STRING data, the text strings are joined. However, if *In1* to *InN* are STRING data, you must use the instruction in a ladder diagram.
The following example is for when *In1* is UV, *In2* is WX and *In3* is YZ. The value of STRING variable *abc* will be 'UVWXYZ'.

LD





Differences in Specifications between Ladder Diagrams and ST

Specifications of this instruction depend on whether it is used in a ladder diagram or ST. The following table gives the differences in specifications. In ladder diagrams, the specifications of the ADD instruction and the + instruction are exactly the same.

Item	Ladder diagram	ST
Maximum number of values to add	5	2*1
Omitting input parameters for values to add	You can omit everything except for the input parameters connected to *InN*.	You cannot omit any input parameters.
Existence of *EN* and *ENO* variables	Present	None
Number of data processing bits if the values to add are all integer data	8, 16, 32, or 64*2	32 or 64*3
Joining of text strings	Possible.	Not possible.
Errors	An error occurs if the size that results from joining text strings exceeds 1,986 bytes.	None

*1 However, you can use more instructions if you use them as operators in an expression, such as *result := val1 + val2 + val3*;. You can use up to 64 instructions in one expression.

*2 The number of processing bits is aligned with the largest data type of all the values to add. For example, if you add SINT, INT, and DINT data, the data processing bits will be aligned to the size of DINT data, i.e., 32-bit processing is performed.

*3 If there is no LINT or ULINT data in the values to add, 32-bit processing is used. For example, if two SINT values are added, 32-bit processing is used. If there is LINT or ULINT data in the values to add, 64-bit processing is used.

Additional Information

- When you calculate real numbers, use the CheckReal instruction (page 2-237) to see if *Out* is positive infinity, negative infinity, or nonnumeric data.
- Use the CONCAT instruction (page 2-554) to join text strings in structured text.

Precautions for Correct Use

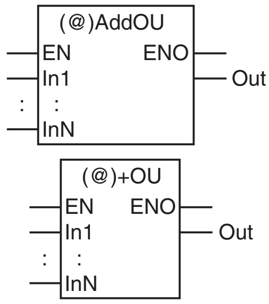
- *Out* can have a different data type than the addition result. However, the data type of *Out* must include the valid range of the data type of the addition result. Otherwise, a building error will occur. Refer to *Data Type Ranking Table* and *Casting Rules* in the *NJ/NX-series CPU Unit Software User's Manual* (Cat. No. W501) or *NY-series Industrial Panel PC / Industrial Box PC Software User's Manual* (Cat. No. W558) for the including relationship on data types.
- When you join text strings, use STRING data for *In1* to *InN* and *Out*.
- An error will not occur even if an underflow or overflow occurs in the addition.
- If an underflow or overflow occurs in addition, the calculation result may not be as expected. Allow sufficient leeway in the sizes of the data types for input and output parameters so that overflows and underflows do not occur.
- Addition results of positive or negative infinity are handled as follows for real number values.

Addition	Addition result
+∞ plus number	+∞
−∞ plus number	−∞
+∞ plus +∞	+∞
−∞ plus −∞	−∞
+∞ plus −∞	Nonnumeric data

- If any of the values of *In1* to *InN* is nonnumeric data, the value of the addition result is nonnumeric data.
- An error will occur in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The size of the joined text string exceeds 1,986 bytes when joining strings.

AddOU (+OU)

The AddOU (+OU) instruction adds integers and real numbers. It also performs an overflow check for integer addition result.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AddOU (+OU)	Addition with Overflow Check	FUN		Out:=AddOU(In1, ···, InN);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1 to InN	Add values	Input	Numbers to add, N = 2 to 5	Depends on data type.	---	0*
Out	Output value	Output	Output value	Depends on data type.	---	---

* If you omit the input parameter that connects to *InN*, the default value is not applied, and a building error will occur. For example, if N is 3 and the input parameters that connect to *In1* and *In2* are omitted, the default values are applied, but if the input parameter that connects to *In3* is omitted, a building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN						OK	OK	OK	OK	OK	OK	OK	OK	OK *	OK *					
Out						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					

* If any of *In1* to *InN* is REAL data, an overflow check is not performed.

Function

The AddOU (+OU) instruction adds between two and five integers or real numbers and outputs the result to output value *Out*.

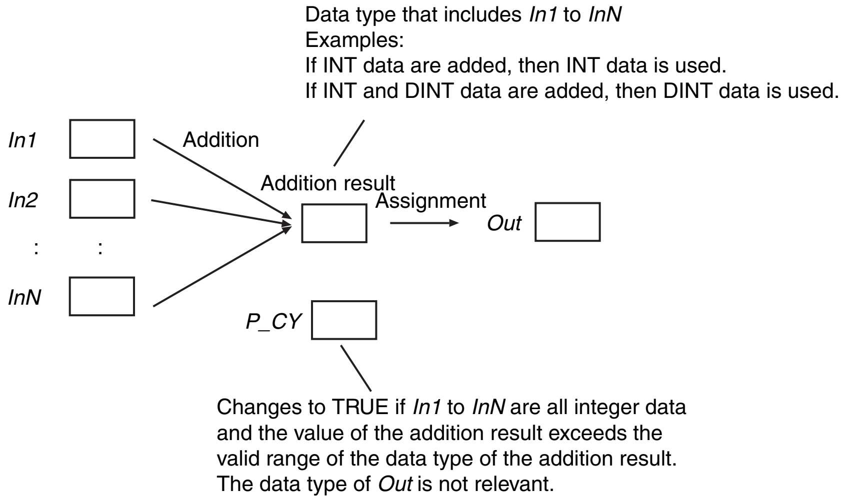
Add values *In1* to *InN* can have different data types. However, use the combination that data types to include are existed. If they are different, calculations are performed with the data type that includes the range of all of the data types. For example, if *In1* is INT data and *In2* is DINT data, calculations are performed with DINT data. Here, the addition result is DINT data.

Refer to *Data Type Ranking Table* and *Casting Rules* in the *NJ/NX-series CPU Unit Software User's Manual* (Cat. No. W501) or *NY-series Industrial Panel PC / Industrial Box PC Software User’s Manual* (Cat. No. W558) for the including relationship on data types.

Processing for Overflows

An overflow occurs if the sum of *In1* to *InN* exceeds the valid range of the data type of the addition result. If all of *In1* to *InN* are integer data and an overflow occurs, the value of the *P_CY* system-defined variable (Carry Flag) changes to TRUE.

If any of *In1* to *InN* is REAL data, an overflow check is not performed. Therefore the value of *P_CY* will not change.



If an overflow occurs, the data types of *In1* to *InN*, the data type of the addition result, the value of the addition result, and the value of *P_CY* will be as shown in the following table.

Data types of *In1* to *InN*	Data type of addition result	Value of addition result	Value of *P_CY*
All integer data	Integer data	Of the sum of *In1* to *InN*, the addition result will be the value that can be expressed by the number of bits in the data type of the addition result.*1	TRUE
At least one real number	Real number data	$\pm\infty$*2	Does not change.

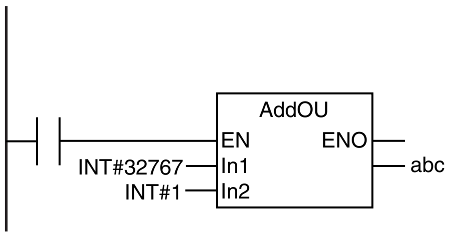
*1 For example, if the value of *In1* is INT#32767 and the value of *In2* is INT#3, the addition result will be INT data. The value of the addition result will be the lower 16 bits of the sum (32,770), i.e., INT#-32766.

*2 If the sum of *In1* to *InN* is positive, the addition result will be positive infinity. If the sum is negative, the addition result will be negative infinity.

Notation Examples

The following example is for when *In1* is INT#32767, *In2* is INT#1 and variable *abc* has an INT data type. *In1* to *InN* are all INT data, so the addition result is INT data. The sum of the two numbers (32,768) exceeds the valid range of INT data, so the value of *P_CY* changes to TRUE. The value of INT variable *abc* will be INT#–32768 (the lower 16 bits of 32,768).

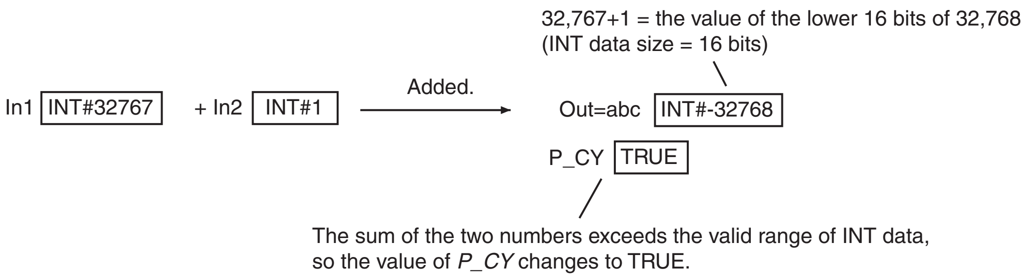
LD



ST

```
abc:=AddOU(INT#32767, INT#1);
```

The AddOU instruction adds *In1* to *InN*.
The sum of the two numbers (32,768) exceeds the valid range of INT data, so the value of *P_CY* changes to TRUE.



Differences in Specifications between Ladder Diagrams and ST

There are no differences in the specifications of this instruction regardless of whether it is used in a ladder diagram or ST. In ladder diagrams, there are no differences in the specifications of the AddOU instruction and the +OU instruction.

Related System-defined Variables

Name	Meaning	Data type	Description
P_CY	Carry (CY) Flag	BOOL	TRUE: Overflow occurred for integer calculations. FALSE: Overflow did not occur for integer calculations.

Additional Information

- If *Out* is REAL data, use the CheckReal instruction (page 2-237) to see if *Out* is positive infinity, negative infinity, or nonnumeric data.
- Use the ADD (+) instruction (page 2-166) if there is no need for an overflow check. It will reduce processing time.

Precautions for Correct Use

- *Out* can have a different data type than the addition result. However, the data type of *Out* must include the valid range of the data type of the addition result. Otherwise, a building error will occur. Refer to *Data Type Ranking Table* and *Casting Rules* in the *NJ/NX-series CPU Unit Software User's Manual* (Cat. No. W501) or *NY-series Industrial Panel PC / Industrial Box PC Software User's Manual* (Cat. No. W558) for the including relationship on data types.
- If an underflow or overflow occurs in addition, the calculation result may not be as expected. Allow sufficient leeway in the sizes of the data types for input and output parameters so that overflows and underflows do not occur.
- Addition results of positive or negative infinity are handled as follows for real number values.

Addition	Addition result
$+\infty$ plus number	$+\infty$
$-\infty$ plus number	$-\infty$
$+\infty$ plus $+\infty$	$+\infty$
$-\infty$ plus $-\infty$	$-\infty$
$+\infty$ plus $-\infty$	Nonnumeric data

- If any of the values of *In1* to *InN* is nonnumeric data, the value of the addition result is nonnumeric data.

SUB (-)

The SUB (-) instruction subtracts integers and real numbers.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SUB (-)	Subtraction	FUN	(@)SUB EN ENO In1 Out In2 (@)- EN ENO In1 Out In2	Out:=In1 - In2;

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Minuend	Input	Minuend	Depends on data type.	---	0*
In2	Subtrahend		Subtrahend			
Out	Output value	Output	Output value	Depends on data type.	---	---

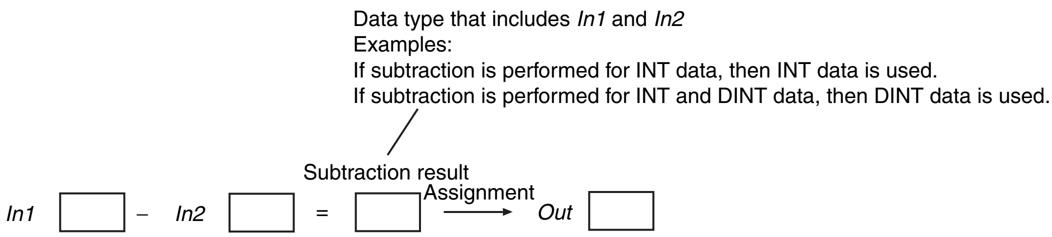
* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit string				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
In2						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Out						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					

Function

The SUB (-) instruction subtracts subtrahend *In2* from minuend *In1* and outputs the result to output value *Out*.

In1 and *In2* can have different data types. However, use the combination that data types to include are existed. If they are different, calculations are performed with the data type that includes the range of all of the data types. For example, if *In1* is INT data and *In2* is DINT data, calculations are performed with DINT data. Here, the subtraction result is DINT data.



Refer to *Data Type Ranking Table* and *Casting Rules* in the *NJ/NX-series CPU Unit Software User's Manual* (Cat. No. W501) or *NY-series Industrial Panel PC / Industrial Box PC Software User's Manual* (Cat. No. W558) for the including relationship on data types.

Processing for Overflows

An overflow occurs if the difference between *In1* and *In2* exceeds the valid range of the data type of the subtraction result. If an overflow occurs, the data types of *In1* and *In2*, the data type of the subtraction result, and the value of the subtraction result will be as shown in the following table.

Data types of *In1* and *In2*	Data type of subtraction result	Value of subtraction result
All integer data	Integer data	Of the difference between *In1* to *InN*, the subtraction result will be the value that can be expressed by the number of bits in the data type of the subtraction result.*1
At least one real number	Real number data	±∞*2

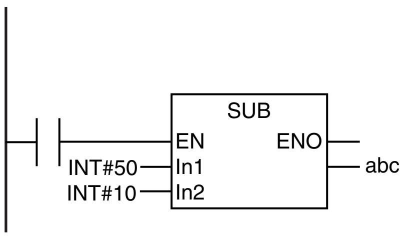
*1 For example, if the value of *In1* is INT#32767 and the value of *In2* is INT#3, the subtraction result will be INT data. The value of the subtraction result will be the lower 16 bits of the difference (32,770), i.e., INT#-32766.

*2 If the difference between *In1* and *InN* is positive, the subtraction result will be positive infinity. If the difference is negative, the subtraction result will be negative infinity.

Notation Examples

The following example is for when *In1* is INT#50 and *In2* is INT#10. The value of INT variable *abc* will be INT#40.

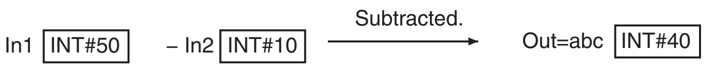
LD



ST

```
abc:=INT#50-INT#10;
```

The SUB instruction subtracts *In2* from *In1*.
The calculation is 50 – 10 = 40, so the value of *abc* will be INT#40.



Differences in Specifications between Ladder Diagrams and ST

Specifications of this instruction depend on whether it is used in a ladder diagram or ST. The following table gives the differences in specifications. In ladder diagrams, the specifications of the SUB instruction and the – instruction are exactly the same.

Item	Ladder diagram	ST
Existence of *EN* and *ENO* variables	Present	None
Number of data processing bits if the minuend and subtrahend are integer data	8, 16, 32, or 64*1	32 or 64*2

*1 The number of processing bits is aligned with the larger data type of the minuend and subtrahend. For example, if you perform subtraction for SINT and DINT data, the data processing bits will be aligned to the size of DINT data, i.e., 32-bit processing is performed.

*2 If there is no LINT or ULINT data in the minuend and subtrahend, 32-bit processing is used. For example, if you perform subtraction for two SINT values, 32-bit processing is used. If there is LINT or ULINT data in the minuend and subtrahend, 64-bit processing is used.

Additional Information

When you calculate real numbers, use the CheckReal instruction (page 2-237) to see if *Out* is positive infinity, negative infinity, or nonnumeric data.

Precautions for Correct Use

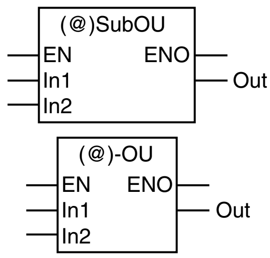
- Out can have a different data type than the subtraction result. However, the data type of *Out* must include the valid range of the data type of the subtraction result. Otherwise, a building error will occur. Refer to *Data Type Ranking Table* and *Casting Rules* in the *NJ/NX-series CPU Unit Software User's Manual* (Cat. No. W501) or *NY-series Industrial Panel PC / Industrial Box PC Software User's Manual* (Cat. No. W558) for the including relationship on data types.
- An error will not occur even if an underflow or overflow occurs in the subtraction.
- If an underflow or overflow occurs in subtraction, the calculation result may not be as expected. Allow sufficient leeway in the sizes of the data types for input and output parameters so that overflows and underflows do not occur.
- Subtraction results of positive or negative infinity are handled as follows for real number values.

Subtraction	Subtraction result
+∞ minus number	+∞
Number minus +∞	−∞
−∞ minus number	−∞
Number minus −∞	+∞
+∞ minus +∞	Nonnumeric data
+∞ minus −∞	+∞
−∞ minus +∞	−∞
−∞ minus −∞	Nonnumeric data

- If the value of *In1* or *In2* is nonnumeric data, the value of the subtraction result is nonnumeric data.

SubOU (-OU)

The SubOU (-OU) instruction subtracts integers or real numbers. It also performs an overflow check for integer subtraction result.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SubOU (-OU)	Subtraction with Overflow Check	FUN		Out:=SubOU(In1, In2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Minuend	Input	Minuend	Depends on data type.	---	0*
In2	Subtrahend		Subtrahend			
Out	Output value	Output	Output value	Depends on data type.	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1						OK	OK	OK	OK	OK	OK	OK	OK	OK *	OK *					
In2						OK	OK	OK	OK	OK	OK	OK	OK	OK *	OK *					
Out						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					

* If *In1* or *In2* is REAL data, an overflow check is not performed.

Function

The SubOU (-OU) instruction subtracts subtrahend In2 from minuend In1 and outputs the result to output value Out.

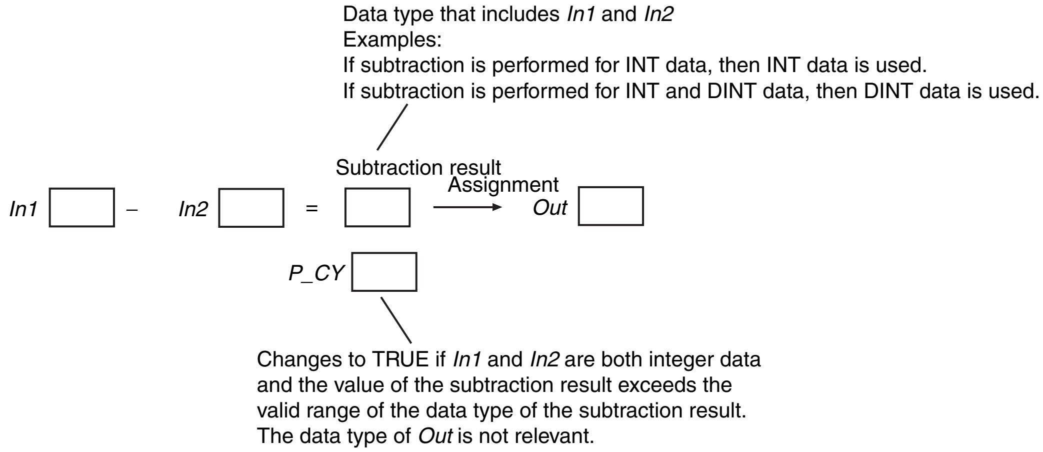
In1 and *In2* can have different data types. However, use the combination that data types to include are existed. If they are different, calculations are performed with the data type that includes the range of all of the data types. For example, if *In1* is INT data and *In2* is DINT data, calculations are performed with DINT data. Here, the subtraction result is DINT data.

Refer to *Data Type Ranking Table* and *Casting Rules* in the *NJ/NX-series CPU Unit Software User's Manual* (Cat. No. W501) or *NY-series Industrial Panel PC / Industrial Box PC Software User’s Manual* (Cat. No. W558) for the including relationship on data types.

Processing for Overflows

An overflow occurs if the difference between *In1* and *In2* exceeds the valid range of the data type of the subtraction result. If *In1* and *In2* are both integer data and an overflow occurs, the value of the *P_CY* system-defined variable (Carry Flag) changes to TRUE.

If either *In1* or *In2* is REAL data, an overflow check is not performed. Therefore the value of *P_CY* will not change.



If an overflow occurs, the data types of *In1* and *In2*, the data type of the subtraction result, the value of the subtraction result, and the value of *P_CY* will be as shown in the following table.

Data types of *In1* and *In2*	Data type of subtraction result	Value of subtraction result	Value of *P_CY*
All integer data	Integer data	Of the difference between *In1* to *InN*, the subtraction result will be the value that can be expressed by the number of bits in the data type of the subtraction result.*1	TRUE
At least one real number	Real number data	±∞*2	Does not change.

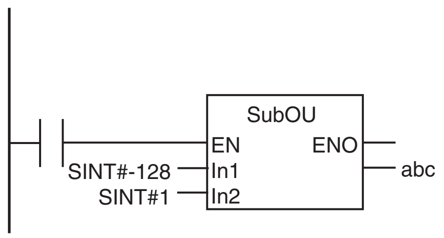
*1 For example, if the value of *In1* is INT#32767 and the value of *In2* is INT#–3, the subtraction result will be INT data. The value of the subtraction result will be the lower 16 bits of the difference (32,770), i.e., INT#-32766.

*2 If the difference between *In1* and *InN* is positive, the subtraction result will be positive infinity. If the difference is negative, the subtraction result will be negative infinity.

Notation Examples

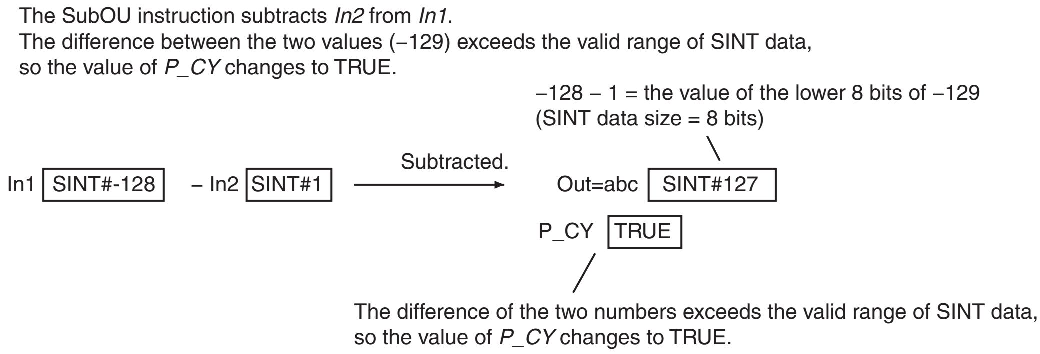
The following example is for when *In1* is SINT#-128, *In2* is SINT#1, and variable *abc* is SINT data. *In1* and *In2* are both SINT data, so the subtraction result is SINT data. The difference between the two values (–129) exceeds the valid range of SINT data, so the value of *P_CY* changes to TRUE. The value of SINT variable *abc* will be SINT#127 (the value of the lower eight bits of –129).

LD



ST

```
abc:=SubOU(SINT#-128, SINT#1);
```



Differences in Specifications between Ladder Diagrams and ST

There are no differences in the specifications of this instruction regardless of whether it is used in a ladder diagram or ST. In ladder diagrams, there are no differences in the specifications of the SubOU instruction and the - OU instruction.

Related System-defined Variables

Name	Meaning	Data type	Description
P_CY	Carry (CY) Flag	BOOL	TRUE: Overflow occurred for integer calculations. FALSE: Overflow did not occur for integer calculations.

Additional Information

- When you calculate real numbers, use the CheckReal instruction (page 2-237) to see if *Out* is positive infinity, negative infinity, or nonnumeric data.
- Use the SUB (–) instruction (page 2-174) if there is no need for an overflow check. It will reduce processing time.

Precautions for Correct Use

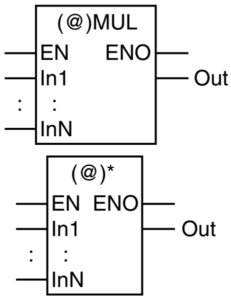
- *Out* can have a different data type than the subtraction result. However, the data type of *Out* must include the valid range of the data type of the subtraction result. Otherwise, a building error will occur. Refer to *Data Type Ranking Table* and *Casting Rules* in the *NJ/NX-series CPU Unit Software User's Manual* (Cat. No. W501) or *NY-series Industrial Panel PC / Industrial Box PC Software User’s Manual* (Cat. No. W558) for the including relationship on data types.
- If an underflow or overflow occurs in subtraction, the calculation result may not be as expected. Allow sufficient leeway in the sizes of the data types for input and output parameters so that overflows and underflows do not occur.
- Subtraction results of positive or negative infinity are handled as follows for real number values.

Subtraction	Subtraction result
$+\infty$ minus number	$+\infty$
Number minus $+\infty$	$-\infty$
$-\infty$ minus number	$-\infty$
Number minus $-\infty$	$+\infty$
$+\infty$ minus $+\infty$	Nonnumeric data
$+\infty$ minus $-\infty$	$+\infty$
$-\infty$ minus $+\infty$	$-\infty$
$-\infty$ minus $-\infty$	Nonnumeric data

- If the value of *In1* or *In2* is nonnumeric data, the value of the subtraction result is nonnumeric data.

MUL (*)

The MUL (*) instruction multiplies integers and real numbers.

Instruction	Name	FB/FUN	Graphic expression	ST expression
MUL (*)	Multiplication	FUN		Out:=In1 * In2;

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1 to InN	Values to multiply	Input	Numbers to multiply Ladder diagram: N = 2 to 5 ST: N = 2*1	Depends on data type.	---	1*2
Out	Output value	Output	Output value	Depends on data type.	---	---

*1 However, you can use more instructions if you use them as operators in an expression, such as *result:= val1 * val2 * val3*;. You can use up to 64 instructions in one expression.

*2 If you omit the input parameter that connects to *InN*, the default value is not applied, and a building error will occur. For example, if N is 3 and the input parameters that connect to *In1* and *In2* are omitted, the default values are applied, but if the input parameter that connects to *In3* is omitted, a building error will occur.

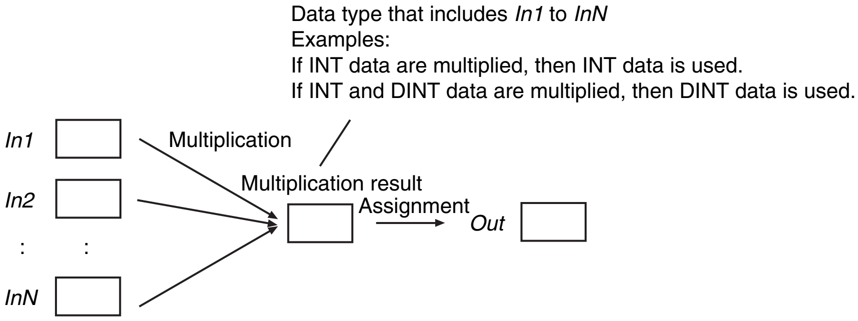
	Boolean	Bit string				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Out						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					

Function

In a ladder diagram, the MUL (*) instruction multiplies between two and five integers or real numbers and outputs the result to output value *Out*. In ST, the MUL instruction multiplies two integers or real numbers and outputs the result to output value *Out*.

Values to multiply *In1* to *InN* can have different data types. However, use the combination that data types to include are existed. If they are different, calculations are performed with the data type that includes the range of all of the data types. For example, if *In1* is INT data and *In2* is DINT data, calculations are performed with DINT data. Here, the multiplication result is DINT data.

Refer to *Data Type Ranking Table* and *Casting Rules* in the *NJ/NX-series CPU Unit Software User's Manual* (Cat. No. W501) or *NY-series Industrial Panel PC / Industrial Box PC Software User's Manual* (Cat. No. W558) for the including relationship on data types.



Processing for Overflows

An overflow occurs if the product of *In1* to *InN* exceeds the valid range of the data type of the multiplication result. If an overflow occurs, the data types of *In1* to *InN*, the data type of the multiplication result, and the value of the multiplication result will be as shown in the following table.

Data types of *In1* to *InN*	Data type of multiplication result	Value of multiplication result
All integer data	Integer data	Of the product of *In1* to *InN*, the multiplication result will be the value that can be expressed by the number of bits in the data type of the multiplication result.*1
At least one real number	Real number data	±∞*2

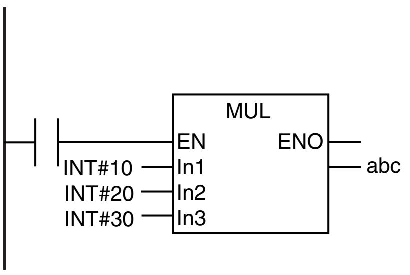
*1 For example, if the value of *In1* is INT#16384 and the value of *In2* is INT#2, the multiplication result will be INT data. The value of the multiplication result will be the lower 16 bits of the product (32,768), i.e., INT#-32768.

*2 If the product of *In1* to *InN* is positive, the multiplication result will be positive infinity. If the product is negative, the multiplication result will be negative infinity.

Notation Examples

The following example is for when *In1* is INT#10, *In2* is INT#20 and *In3* is INT#30. The value of INT variable *abc* will be INT#6000.

LD



ST

```
abc:=INT#10*INT#20*INT#30;
```

The MUL instruction multiplies *In1* to *InN*.
The calculation is 10 × 20 × 30 = 6,000, so the value of *abc* will be INT#6000.

Differences in Specifications between Ladder Diagrams and ST

Specifications of this instruction depend on whether it is used in a ladder diagram or ST. The following table gives the differences in specifications. In ladder diagrams, the specifications of the MUL instruction and the * instruction are exactly the same.

Item	Ladder diagram	ST
Maximum number of values to multiply	5	2*1
Omitting input parameters for values to multiply	You can omit everything except for the input parameters connected to *InN*.	You cannot omit any input parameters.
Existence of *EN* and *ENO* variables	Present	None
Number of data processing bits if the values to multiple are all integer data	8, 16, 32, or 64*2	32 or 64*3

*1 However, you can use more instructions if you use them as operators in an expression, such as *result:= val1 * val2 * val3*;. You can use up to 64 instructions in one expression.

*2 The number of processing bits is aligned with the largest data type of all the values to multiply. For example, if you multiply SINT, INT, and DINT data, the data processing bits will be aligned to the size of DINT data, i.e., 32-bit processing is performed.

*3 If there is no LINT or ULINT data in the values to multiply, 32-bit processing is used. For example, if two SINT values are multiplied, 32-bit processing is used. If there is LINT or ULINT data in the values to multiply, 64-bit processing is used.

Additional Information

When you calculate real numbers, use the CheckReal instruction (page 2-237) to see if *Out* is positive infinity, negative infinity, or nonnumeric data.

Precautions for Correct Use

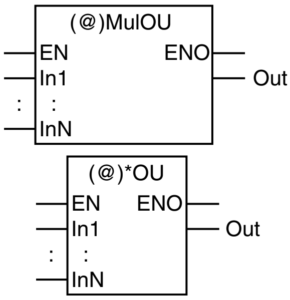
- *Out* can have a different data type than the multiplication result. However, the data type of *Out* must include the valid range of the data type of the multiplication result. Otherwise, a building error will occur. Refer to *Data Type Ranking Table* and *Casting Rules* in the *NJ/NX-series CPU Unit Software User's Manual* (Cat. No. W501) or *NY-series Industrial Panel PC / Industrial Box PC Software User’s Manual* (Cat. No. W558) for the including relationship on data types.
- An error will not occur even if an underflow or overflow occurs in the multiplication.
- If an underflow or overflow occurs in multiplication, the calculation result may not be as expected. Allow sufficient leeway in the sizes of the data types for input and output parameters so that overflows and underflows do not occur.
- Multiplication results of positive or negative infinity are handled as follows for real number values.

Multiplication	Multiplication result
+∞ times positive number	+∞
+∞ times negative number	−∞
−∞ times positive number	−∞
−∞ times negative number	+∞
+∞ times +∞	+∞
−∞ times −∞	+∞
+∞ times −∞	−∞
+∞ times 0	Nonnumeric data
−∞ times 0	Nonnumeric data

- If any of the values of *In1* to *InN* is nonnumeric data, the value of the multiplication result is nonnumeric data.

MulOU (*OU)

The MulOU (*OU) instruction multiplies integers and real numbers and outputs the result. It also performs an overflow check for integer multiplication result.

Instruction	Name	FB/FUN	Graphic expression	ST expression
MulOU (*OU)	Multiplication with Overflow Check	FUN		Out:=MulOU(In1, ···, InN);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1 to InN	Values to multiply	Input	Numbers to multiply, N = 2 to 5	Depends on data type.	---	1*
Out	Output value	Output	Output value	Depends on data type.	---	---

* If you omit the input parameter that connects to *InN*, the default value is not applied, and a building error will occur. For example, if N is 3 and the input parameters that connect to *In1* and *In2* are omitted, the default values are applied, but if the input parameter that connects to *In3* is omitted, a building error will occur.

	Boolean	Bit string				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN						OK	OK	OK	OK	OK	OK	OK	OK	OK *	OK *					
Out						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					

* If any of *In1* to *InN* is REAL data, an overflow check is not performed.

Function

The MUL (*) instruction multiplies between two and five integers or real numbers and outputs the result to output value *Out*.

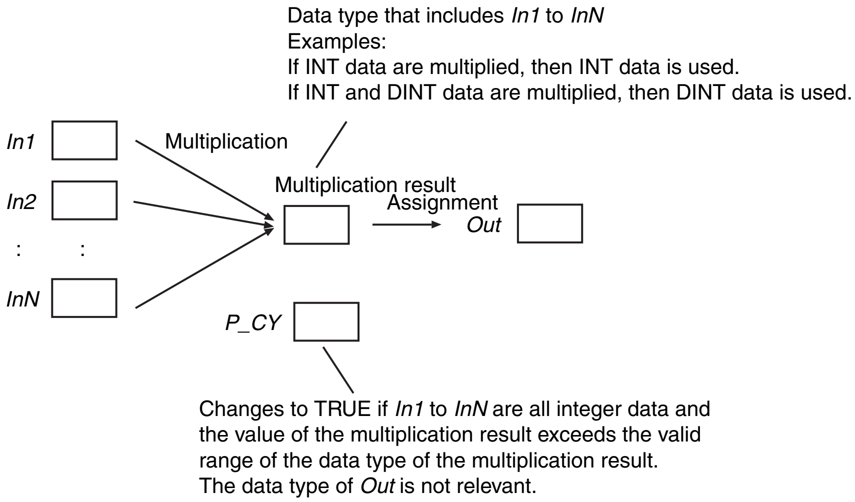
Values to multiply *In1* to *InN* can have different data types. However, use the combination that data types to include are existed. If they are different, calculations are performed with the data type that includes the range of all of the data types. For example, if *In1* is INT data and *In2* is DINT data, calculations are performed with DINT data. Here, the multiplication result is DINT data.

Refer to *Data Type Ranking Table* and *Casting Rules* in the *NJ/NX-series CPU Unit Software User's Manual* (Cat. No. W501) or *NY-series Industrial Panel PC / Industrial Box PC Software User’s Manual* (Cat. No. W558) for the including relationship on data types.

Processing for Overflows

An overflow occurs if the product of *In1* to *InN* exceeds the valid range of the data type of the multiplication result. If all of *In1* to *InN* are integer data and an overflow occurs, the value of the *P_CY* system-defined variable (Carry Flag) changes to TRUE.

If any of *In1* to *InN* is REAL data, an overflow check is not performed. Therefore the value of *P_CY* will not change.



If an overflow occurs, the data types of *In1* to *InN*, the data type of the multiplication result, the value of the multiplication result, and the value of *P_CY* will be as shown in the following table.

Data types of *In1* to *InN*	Data type of multiplication result	Value of multiplication result	Value of *P_CY*
All integer data	Integer data	Of the product of *In1* to *InN*, the multiplication result will be the value that can be expressed by the number of bits in the data type of the multiplication result.*1	TRUE
At least one real number	Real number data	±∞*2	Does not change.

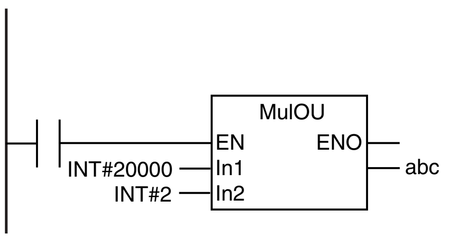
*1 For example, if the value of *In1* is INT#16384 and the value of *In2* is INT#2, the multiplication result will be INT data. The value of the multiplication result will be the lower 16 bits of the product (32,768), i.e., INT#-32768.

*2 If the product of *In1* to *InN* is positive, the multiplication result will be positive infinity. If the product is negative, the multiplication result will be negative infinity.

Notation Examples

The following example is for when *In1* is INT#20000, *In2* is INT#2 and variable *abc* has an INT data type. *In1* to *InN* are all INT data, so the addition result is INT data. The product of the two values (40,000) exceeds the valid range of INT data, so the value of *P_CY* changes to TRUE. The value of INT variable *abc* will be INT#-25536 (the lower 16 bits of 40,000).

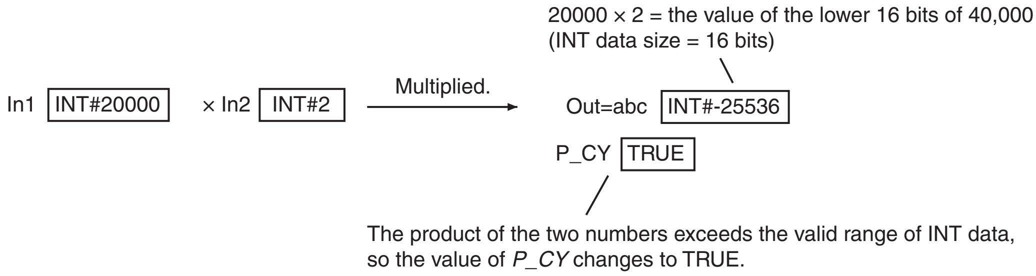
LD



ST

```
abc:=MulOU(INT#20000, INT#2);
```

The MulOU instruction multiplies *In1* to *InN*.
The product of the two values (40,000) exceeds the valid range of INT data, so the value of *P_CY* changes to TRUE.



Differences in Specifications between Ladder Diagrams and ST

There are no differences in the specifications of this instruction regardless of whether it is used in a ladder diagram or ST. In ladder diagrams, there are no differences in the specifications of the MulOU instruction and the *OU instruction.

Related System-defined Variables

Name	Meaning	Data type	Description
P_CY	Carry (CY) Flag	BOOL	TRUE: Overflow occurred for integer calculations. FALSE: Overflow did not occur for integer calculations.

Additional Information

- When you calculate real numbers, use the CheckReal instruction (page 2-237) to see if *Out* is positive infinity, negative infinity, or nonnumeric data.
- Use the MUL (*) instruction (page 2-181) if there is no need for an overflow check. It will reduce processing time.

Precautions for Correct Use

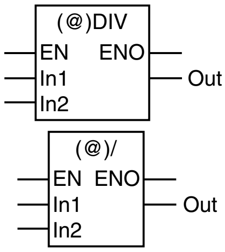
- *Out* can have a different data type than the multiplication result. However, the data type of *Out* must include the valid range of the data type of the multiplication result. Otherwise, a building error will occur. Refer to *Data Type Ranking Table* and *Casting Rules* in the *NJ/NX-series CPU Unit Software User's Manual* (Cat. No. W501) or *NY-series Industrial Panel PC / Industrial Box PC Software User's Manual* (Cat. No. W558) for the including relationship on data types.
- If an underflow or overflow occurs in multiplication, the calculation result may not be as expected. Allow sufficient leeway in the sizes of the data types for input and output parameters so that overflows and underflows do not occur.
- Multiplication results of positive or negative infinity are handled as follows for real number values.

Multiplication	Multiplication result
+∞ times positive number	+∞
+∞ times negative number	−∞
−∞ times positive number	−∞
−∞ times negative number	+∞
+∞ times +∞	+∞
−∞ times −∞	+∞
+∞ times −∞	−∞
+∞ times 0	Nonnumeric data
−∞ times 0	Nonnumeric data

- If any of the values of *In1* to *InN* is nonnumeric data, the value of the multiplication result is nonnumeric data.

DIV (/)

The DIV (/) instruction divides integers or real numbers.

Instruction	Name	FB/FUN	Graphic expression	ST expression
DIV (/)	Division	FUN		Out:=In1/ In2;

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Dividend	Input	Dividend	Depends on data type.	---	*
In2	Divisor		Divisor			
Out	Output value	Output	Output value	Depends on data type.	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit string				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
In2						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Out						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					

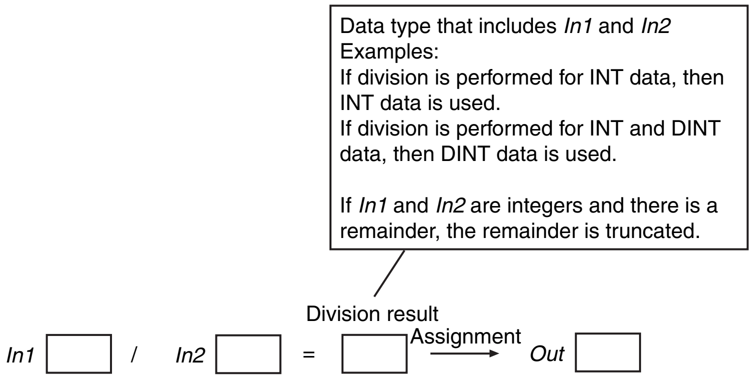
Function

The DIV (/) instruction divides dividend *In1* by divisor *In2* and outputs the result to output value *Out*.

In1 and *In2* can have different data types. However, use the combination that data types to include are existed. If they are different, calculations are performed with the data type that includes the range of all of the data types. For example, if *In1* is INT data and *In2* is DINT data, calculations are performed with DINT data. Here, the division result is DINT data.

If *In1* and *In2* are integers and there is a remainder, the remainder is truncated.

Refer to *Data Type Ranking Table* and *Casting Rules* in the *NJ/NX-series CPU Unit Software User's Manual* (Cat. No. W501) or *NY-series Industrial Panel PC / Industrial Box PC Software User's Manual* (Cat. No. W558) for the including relationship on data types.



Processing for Overflows

An overflow occurs if the quotient of *In1* and *In2* exceeds the valid range of the data type of the division result. If an overflow occurs, the data types of *In1* and *In2*, the data type of the division result, and the value of the division result will be as shown in the following table.

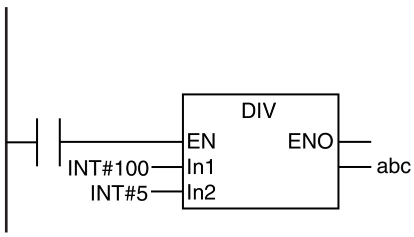
Data types of *In1* and *In2*	Data type of division result	Value of division result
At least one real number	Real number data	±∞*

* If the quotient of *In1* and *In2* is positive, the division result will be positive infinity. If the quotient is negative, the division result will be negative infinity.

Notation Examples

The following example is for when *In1* is INT#100 and *In2* is INT#5. The value of INT variable *abc* will be INT#20.

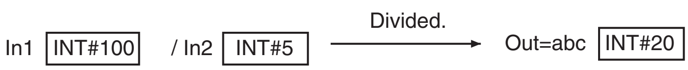
LD



ST

```
abc:=INT#100/INT#5;
```

The DIV instruction divides *In1* by *InN*.
The calculation is 100/5 = 20, so the value of *abc* will be INT#20.



Differences in Specifications between Ladder Diagrams and ST

Specifications of this instruction depend on whether it is used in a ladder diagram or ST. The following table gives the differences in specifications. In ladder diagrams, the specifications of the DIV instruction and the / instruction are exactly the same.

Item	Ladder diagram	ST
Existence of *EN* and *ENO* variables	Present	None
Number of data processing bits if the dividend and divisor are integer data	8, 16, 32, or 64*1	32 or 64*2

*1 The number of processing bits is aligned with the larger data type of the dividend and divisor. For example, if you perform division for SINT and DINT data, the data processing bits will be aligned to the size of DINT data, i.e., 32-bit processing is performed.

*2 If there is no LINT or ULINT data in the dividend and divisor, 32-bit processing is used. For example, if you perform division for two SINT values, 32-bit processing is used. If there is LINT or ULINT data in the dividend and divisor, 64-bit processing is used.

Additional Information

When you calculate real numbers, use the CheckReal instruction (page 2-237) to see if *Out* is positive infinity, negative infinity, or nonnumeric data.

Precautions for Correct Use

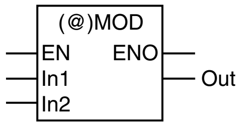
- *Out* can have a different data type than the division result. However, the data type of *Out* must include the valid range of the data type of the division result. Otherwise, a building error will occur. Refer to *Data Type Ranking Table* and *Casting Rules* in the *NJ/NX-series CPU Unit Software User's Manual* (Cat. No. W501) or *NY-series Industrial Panel PC / Industrial Box PC Software User's Manual* (Cat. No. W558) for the including relationship on data types.
- An error will not occur even if an underflow or overflow occurs in the division.
- If an underflow or overflow occurs in division, the calculation result may not be as expected. Allow sufficient leeway in the sizes of the data types for input and output parameters so that overflows and underflows do not occur.
- Division results of positive infinity, negative infinity, or 0 are handled as follows for real number values.

		In1				
		+∞	Positive number	0	Negative number	−∞
In2	+∞	Nonnumeric data	0	0	0	Nonnumeric data
	Positive number	+∞	Positive number	0	Negative number	−∞
	0	+∞	+∞	Nonnumeric data	−∞	−∞
	Negative number	−∞	Negative number	0	Positive number	+∞
	−∞	Nonnumeric data	0	0	0	Nonnumeric data

- If the value of *In1* or *In2* is nonnumeric data, the value of the division result is nonnumeric data.
- An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.
  - *In1* and *In2* are integers and the value of *In2* is 0.

MOD

The MOD instruction finds the remainder for division of integers.

Instruction	Name	FB/FUN	Graphic expression	ST expression
MOD	Modulo-division	FUN		Out:=In1 MOD In2;

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Dividend	Input	Dividend	Depends on data type.	---	*
In2	Divisor		Divisor			
Out	Remainder	Output	Remainder	Depends on data type.	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1						OK	OK	OK	OK	OK	OK	OK	OK							
In2						OK	OK	OK	OK	OK	OK	OK	OK							
Out						OK	OK	OK	OK	OK	OK	OK	OK							

Function

The MOD instruction divides dividend *In1* by divisor *In2* to find the remainder.

The data types of *In1*, *In2*, and remainder *Out* can have different data types. However, use the combination that data types to include are existed.

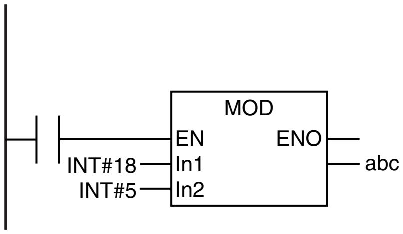
Refer to *Data Type Ranking Table* and *Casting Rules* in the *NJ/NX-series CPU Unit Software User's Manual* (Cat. No. W501) or *NY-series Industrial Panel PC / Industrial Box PC Software User’s Manual* (Cat. No. W558) for the including relationship on data types.

This instruction performs the calculation with the following formula.
Out = In1 – (In1/In2)* In2 (The decimal point is truncated in the division operation.)
Examples of the values of *In1*, *In2*, and *Out* are given in the following table.

Value of *In1*	Value of *In2*	Value of *Out*
5	3	2
5	–3	2
–5	3	–2
–5	–3	–2

The following example is for when *In1* is INT#18 and *In2* is INT#5. The value of variable *abc* will be INT#3.

LD



ST

```
abc:=INT#18 MOD INT#5;
```

The MOD instruction divides *In1* by *In2* to find the remainder.
The remainder of 18/5 is 3, so the value of *abc* will be INT#3.

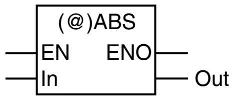
In1 $\boxed{\text{INT\#18}}$ / In2 $\boxed{\text{INT\#5}}$ $\xrightarrow{\text{Remainder calculated.}}$ Out=abc $\boxed{\text{INT\#3}}$

Precautions for Correct Use

- Set the data type of *Out* to include the valid ranges of *In1* and *In2*. Refer to *Data Type Ranking Table* and *Casting Rules* in the *NJ/NX-series CPU Unit Software User's Manual* (Cat. No. W501) or *NY-series Industrial Panel PC / Industrial Box PC Software User's Manual* (Cat. No. W558) for the including relationship on data types.

ABS

The ABS instruction finds the absolute value of an integer or real number.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ABS	Absolute Value	FUN		Out:=ABS(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Number to process	Input	Number to process	Depends on data type.	---	*1
Out	Absolute value	Output	Absolute value	Depends on data type. *2	---	---

*1 If you omit an input parameter, the default value is not applied. A building error will occur.

*2 Negative numbers are excluded.

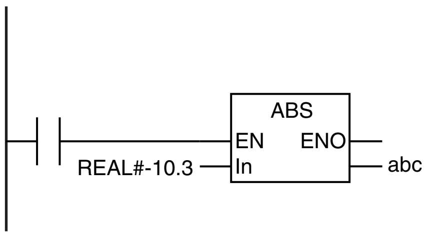
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Out						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					

Function

The ABS instruction outputs the absolute value of the number to process *In*. The data types of *In* and absolute value *Out* can have different data types.

The following example is for when *In* is REAL#–10.3. The value of variable *abc* will be REAL#10.3.

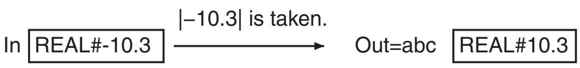
LD



ST

```
abc:=ABS(REAL#-10.3);
```

The ABS instruction outputs the absolute value of *In*.
The absolute value of REAL#–10.3 is found, so the value of *abc* will be REAL#10.3.



Additional Information

When you calculate real numbers, use the CheckReal instruction (page 2-237) to see if *Out* is positive infinity, negative infinity, or nonnumeric data.

Precautions for Correct Use

- Set the data type of *Out* to include the absolute value of *In*.
- If the value of *In* is positive infinity, negative infinity, or nonnumeric data, the value of *Out* is as shown below.

Value of *In*	Value of *Out*
$+\infty$	$+\infty$
$-\infty$	$+\infty$
Nonnumeric data	Nonnumeric data

RadToDeg and DegToRad

RadToDeg: Converts a real number from radians (rad) to degrees (°).

DegToRad: Converts a real number from degrees (°) to radians (rad).

Instruction	Name	FB/FUN	Graphic expression	ST expression
RadToDeg	Radians to Degrees	FUN	(@)RadToDeg EN ENO In Out	Out:=RadToDeg(In);
DegToRad	Degrees to Radians	FUN	(@)DegToRad EN ENO In Out	Out:=DegToRad(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	Depends on data type.	• RadToDeg: Radians • DegToRad: Degrees	*
Out	Conversion result	Output	Conversion result	Depends on data type.	• RadToDeg: Degrees • DegToRad: Radians	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK	OK					
Out														OK	OK					

Function

● RadToDeg

The RadToDeg instruction converts the data to convert *In* from radians (rad) to degrees (°). The following conversion is used.

Out=In*180/π

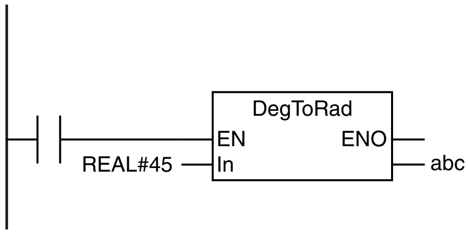
● DegToRad

The DegToRad instruction converts the data to convert *In* from degrees (°) to radians (rad). The following conversion is used.

Out=In*π/180

The following example for the DegToRad instruction is for when *In* is REAL#45. The value of the REAL variable *abc* will be REAL#0.785398.

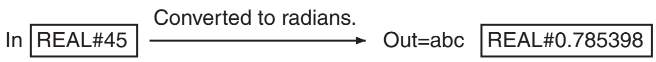
LD



ST

```
abc:=DegToRad(REAL#45);
```

The DegToRad instruction converts the value of *In* from degrees (°) to radians (rad).
An angle of 45° is 0.785398 rad, so the value of *abc* will be REAL#0.785398.



Additional Information

Use the CheckReal instruction (page 2-237) to see if *Out* is positive infinity, negative infinity, or nonnumeric data.

Precautions for Correct Use

- If the absolute value of the conversion result exceeds the maximum value of the data type of *Out*, the value of *Out* will be positive or negative infinity.
- If the absolute value of the conversion result is lower than the minimum value of the data type of *Out*, the value of *Out* will be 0.
- Make sure that the data type of *Out* is equal to or larger than the data type of *In*.
- If the value of *In* is positive infinity, negative infinity, or nonnumeric data, the value of *Out* is as shown below.

Value of *In*	Value of *Out*
+∞	+∞
−∞	−∞
Nonnumeric data	Nonnumeric data

- If you pass an integer parameter to *In*, the data type is converted as follows:

Data type of parameter that is passed to *In*	Data type of *In*
USINT, UINT, SINT, or INT	REAL
UDINT or DINT	LREAL
ULINT or LINT	A building error will occur.

SIN, COS, and TAN

These instructions perform trigonometric calculations on real numbers.

SIN: Finds the sine of a number.

COS: Finds the cosine of a number.

TAN: Finds the tangent of a number.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SIN	Sine in Radians	FUN	(@)SIN EN ENO In Out	Out:=SIN(In);
COS	Cosine in Radians	FUN	(@)COS EN ENO In Out	Out:=COS(In);
TAN	Tangent in Radians	FUN	(@)TAN EN ENO In Out	Out:=TAN(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Number to process	Input	Number to process	Depends on data type.	Radians	*1
Out	Calculation result	Output	Calculation result	• SIN*2 • COS*2 • TAN Depends on data type.	---	---

*1 If you omit an input parameter, the default value is not applied. A building error will occur.

*2 The valid range is –1.000000e+0 to 1.000000e+0 for REAL data. The valid range is –1.00000000000000e+0 to 1.00000000000000e+0 for LREAL data.

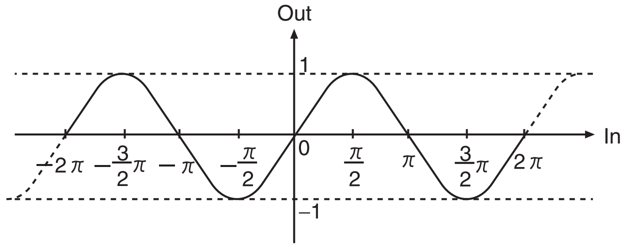
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK	OK					
Out														OK	OK					

Function

These instructions perform trigonometric calculations on real numbers. Number to process *In* is an angle in radians (rad).

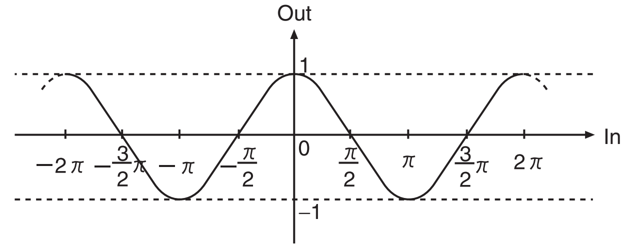
● SIN

The SIN instruction finds the sine of *In.*



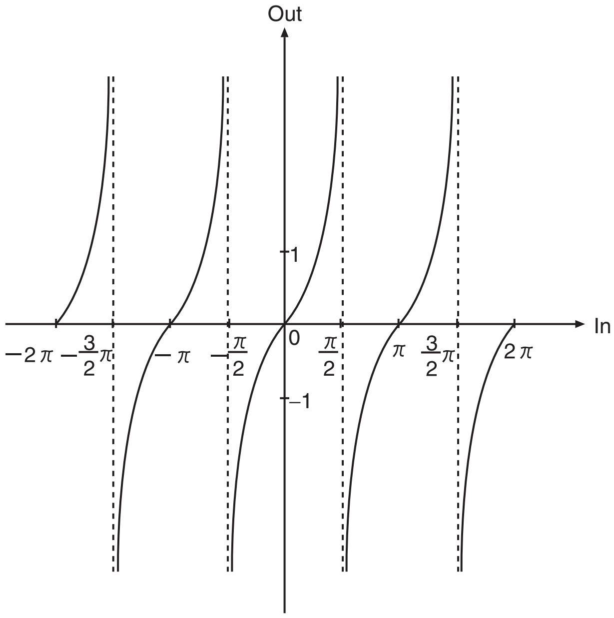
● COS

The COS instruction finds the cosine of *In.*



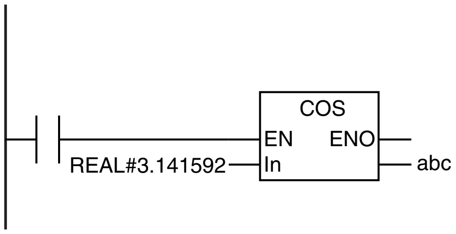
● TAN

The TAN instruction finds the tangent of *In.*



The following example for the COS instruction is for when *In* is REAL#3.141592. The value of variable *abc* will be REAL#–1.0.

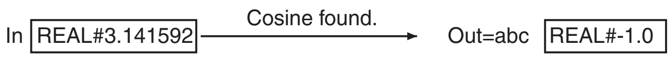
LD



ST

```
abc:=COS(REAL#3.141592);
```

The COS instruction finds the cosine of *In*.
The cosine of 3.141592 is –1.0, so the value of *abc* will be REAL#–1.0.



Additional Information

- Use the RadToDeg and DegToRad instructions (page 2-196) to convert between degrees and radians.
- If *In* for the TAN instruction is nπ/2, when n is an integer, then the value of *Out* will be positive or negative infinity. Use the CheckReal instruction (page 2-237) to see if the value of *Out* is positive infinity or negative infinity.

Precautions for Correct Use

- If the value of *In* is positive infinity, negative infinity, or nonnumeric data, the value of *Out* is nonnumeric data.
- If you pass an integer parameter to *In*, the data type is converted as follows:

Data type of parameter that is passed to *In*	Data type of *In*
USINT, UINT, SINT, or INT	REAL
UDINT or DINT	LREAL
ULINT or LINT	A building error will occur.

ASIN, ACOS, and ATAN

These instructions perform inverse trigonometric calculations on real numbers.

ASIN: Finds the arc sine of a number.

ACOS: Finds the arc cosine of a number.

ATAN: Finds the arc tangent of a number.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ASIN	Principal Arc Sine	FUN	(@)ASIN EN ENO In Out	Out:=ASIN(In);
ACOS	Principal Arc Cosine	FUN	(@)ACOS EN ENO In Out	Out:=ACOS(In);
ATAN	Principal Arc Tangent	FUN	(@)ATAN EN ENO In Out	Out:=ATAN(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Number to process	Input	Number to process	Depends on data type.	---	*
Out	Calculation result	Output	Calculation result	• ASIN $-\pi/2$ to $\pi/2$ • ACOS 0 to π • ATAN $-\pi/2$ to $\pi/2$	rad	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

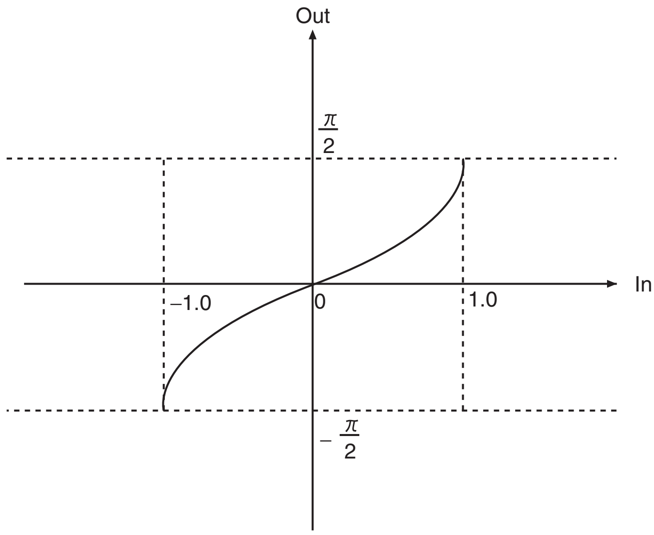
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK	OK					
Out														OK	OK					

Function

These instructions perform inverse trigonometric calculations on real numbers. The calculation result *Out* is an angle in radians (rad).

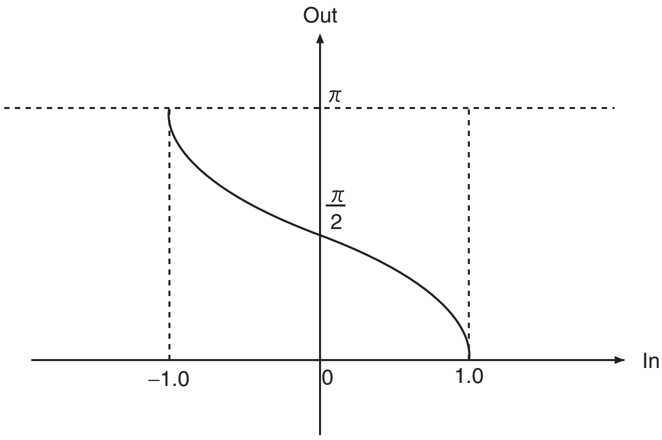
● ASIN

The ASIN instruction finds the arc sine of *In*. *Out* is between $-\pi/2$ and $\pi/2$.



ACOS

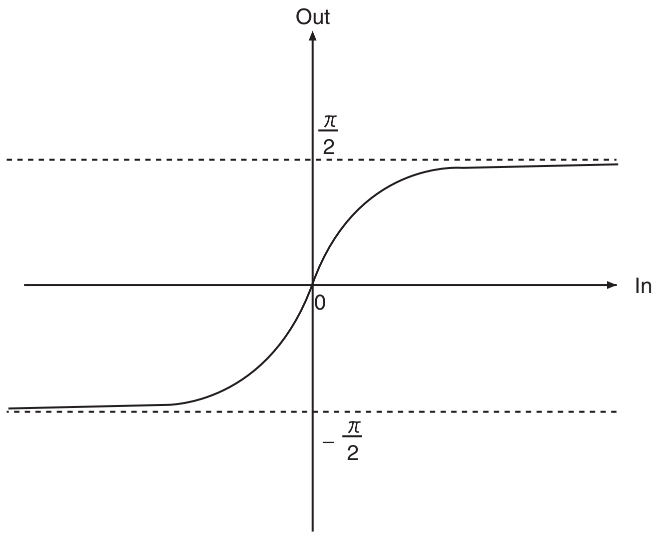
The ACOS instruction finds the arc cosine of *In*. *Out* is between 0 and π.



ATAN

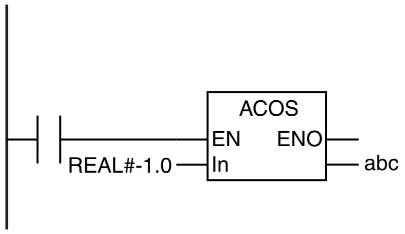
The ATAN instruction finds the arc tangent of *In*. *Out* is between −π/2 and π/2.

If the value of *In* is positive infinity, the value of *Out* is π/2. If the value of *In* is negative infinity, the value of *Out* is −π/2.



The following example for the ACOS instruction is for when *In* is REAL#–1.0. The value of variable *abc* will be REAL#3.141592.

LD



ST

```
abc:=ACOS(REAL#-1.0);
```

The ACOS instruction finds the arccosine of *In*.
The arccosine of –1.0 is 3.141592, so the value of *abc* will be REAL#3.141592.

In $\boxed{\text{REAL\#-1.0}}$ $\xrightarrow{\text{Arccosine found.}}$ Out=abc $\boxed{\text{REAL\#3.141592}}$

Additional Information

Use the RadToDeg and DegToRad instructions (page 2-196) to convert between degrees and radians.

Precautions for Correct Use

- If *In* is not between –1.0 and 1.0 for the ASIN or ACOS instruction, the value of *Out* is nonnumeric data. That also applies when the value of *In* is negative infinity, positive infinity, or nonnumeric data.
- If the value of *In* is nonnumeric data for the ATAN instruction, the value of *Out* is nonnumeric data.
- If you pass an integer parameter to *In*, the data type is converted as follows:

Data type of parameter that is passed to *In*	Data type of *In*
USINT, UINT, SINT, or INT	REAL
UDINT or DINT	LREAL
ULINT or LINT	A building error will occur.

SQRT

The SQRT instruction finds the square root of a number.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SQRT	Square Root	FUN	(@)SQRT EN ENO In Out	Out:=SQRT(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Number to process	Input	Number to process	Depends on data type. *1	---	*2
Out	Square root	Output	Square root	*3	---	---

*1 Negative numbers are excluded.

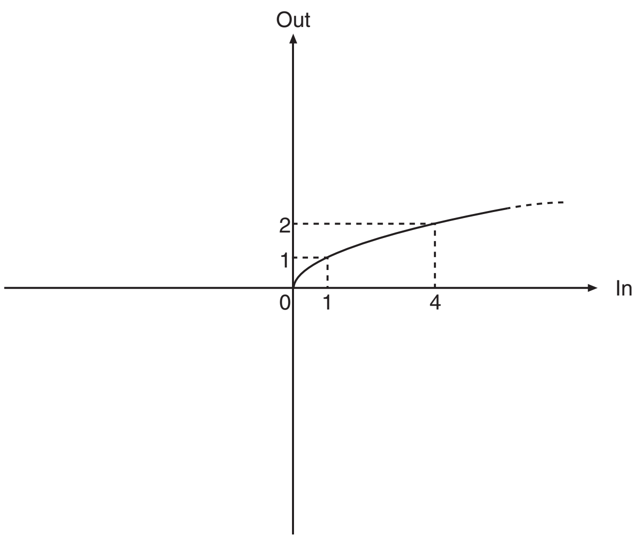
*2 If you omit an input parameter, the default value is not applied. A building error will occur.

*3 The valid range is 0.000000e+00 to 1.844674e+19 or positive infinity for REAL data. The valid range is 0.00000000000000e+000 to 1.34078079299425e+154 or positive infinity for LREAL data.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK	OK					
Out														OK	OK					

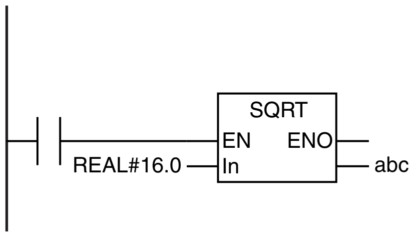
Function

The SQRT instruction finds the square root of number to process *In*. The data types of *In* and square root *Out* can have different data types.



The following example is for when *In* is REAL#16.0. The value of variable *abc* will be REAL#4.0.

LD



ST

```
abc:=SQRT(REAL#16.0);
```

The SQRT instruction finds the square root of *In*.
The square root of 16.0 is 4.0, so the value of *abc* will be REAL#4.0.

In $\boxed{\text{REAL\#16.0}}$ $\xrightarrow{\text{Square root is found.}}$ Out=abc $\boxed{\text{REAL\#4.0}}$

Additional Information

Use the CheckReal instruction (page 2-237) to see if the value of *Out* is positive infinity.

Precautions for Correct Use

- If the value of *In* is not a positive number, the value of *Out* is as shown below.

Value of *In*	Value of *Out*
Negative number	Nonnumeric data
0	0
+∞	+∞
−∞	Nonnumeric data
Nonnumeric data	Nonnumeric data

- If you pass an integer parameter to In, the data type is converted as follows:

Data type of parameter that is passed to *In*	Data type of *In*
USINT, UINT, SINT, or INT	REAL
UDINT or DINT	LREAL
ULINT or LINT	A building error will occur.

LN and LOG

These instructions find the logarithm of a real number.

LN: Finds the natural logarithm of a number.

LOG: Finds the base-10 logarithm of a number.

Instruction	Name	FB/FUN	Graphic expression	ST expression
LN	Natural Logarithm	FUN	(@)LN EN ENO In Out	Out:=LN(In);
LOG	Logarithm Base 10	FUN	(@)LOG EN ENO In Out	Out:=LOG(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Number to process	Input	Number to process	Depends on data type. *1	---	*2
Out	Logarithm	Output	Logarithm	*3	---	---

*1 Negative numbers are excluded.

*2 If you omit an input parameter, the default value is not applied. A building error will occur.

*3 LN:
If *In* and *Out* are REAL data: –8.73365448e+1 to 8.87228390e+1, or –∞/+∞
If *In* is REAL and *Out* is LREAL data: –8.7336544750000000e+1 to 8.8722839050000000e+1 or –∞/+∞
If *In* is LREAL and *Out* is REAL data: –7.08384950e+2 to 7.09782712e+2 or –∞/+∞
If *In* and *Out* are LREAL data: –7.0838495021978327e+1 to 7.0978271289338399e+2 or –∞/+∞

LOG:
If *In* and *Out* are REAL data: –3.79297795e+1 to 3.85318394e+1 or –∞/+∞
If *In* is REAL and *Out* is LREAL data: –3.7929779453965430e+1 to 3.8531839419564961e+1 or –∞/+∞
If *In* is LREAL and *Out* is REAL data: –3.07652656e+2 to 3.08254716e+2 or –∞/+∞
If *In* and *Out* are LREAL data: –3.0765265556858878e+2 to 3.0825471555991674e+2 or –∞/+∞

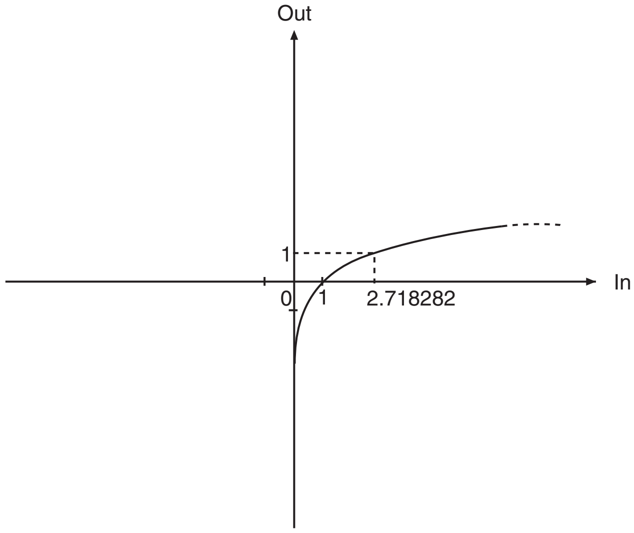
	Boolean	Bit string				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK	OK					
Out														OK	OK					

Function

These instructions find the logarithm of a real number.

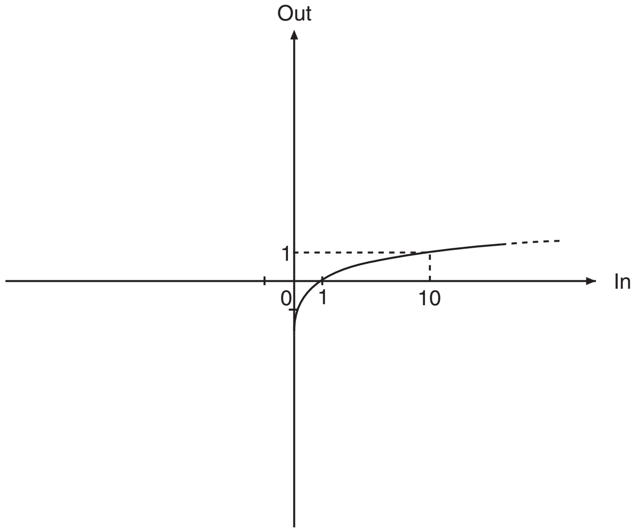
● LN

The LN instruction finds the natural logarithm (logarithm to base e, where e = 2.718282).



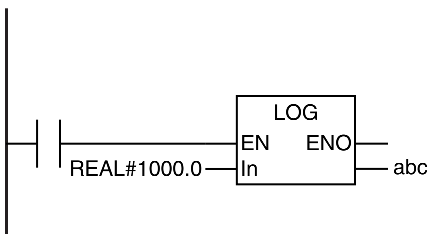
● LOG

The LOG instruction finds the base-10 logarithm.



The following example for the LOG instruction is for when *In* is REAL#1000.0. The value of variable *abc* will be REAL#3.0.

LD



ST

```
abc:=LOG(REAL#1000.0);
```

The LOG instruction finds the base-10 logarithm of a real number.
The base-10 logarithm of 1,000.0 is 3.0 so the value of *abc* will be REAL#3.0.

In $\boxed{\text{REAL\#1000.0}} \xrightarrow{\text{Common logarithm is taken.}}$ Out=abc $\boxed{\text{REAL\#3.0}}$

Additional Information

Use the CheckReal instruction (page 2-237) to see if *Out* is positive infinity, negative infinity, or nonnumeric data.

Precautions for Correct Use

- If the value of *In* is not a positive number, the value of *Out* is as shown below.

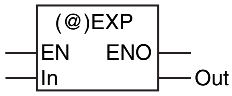
Value of *In*	Value of *Out*
Negative number	Nonnumeric data
0	−∞
+∞	+∞
−∞	Nonnumeric data
Nonnumeric data	Nonnumeric data

- If you pass an integer parameter to *In*, the data type is converted as follows:

Data type of parameter that is passed to *In*	Data type of *In*
USINT, UINT, SINT, or INT	REAL
UDINT or DINT	LREAL
ULINT or LINT	A building error will occur.

EXP

The EXP instruction performs calculations for the natural exponential function.

Instruction	Name	FB/FUN	Graphic expression	ST expression
EXP	Natural Exponential Operation	FUN		Out:=EXP(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Exponent	Input	Exponent	Depends on data type.	---	*1
Out	Calculation result	Output	Calculation result	Depends on data type. *2	---	---

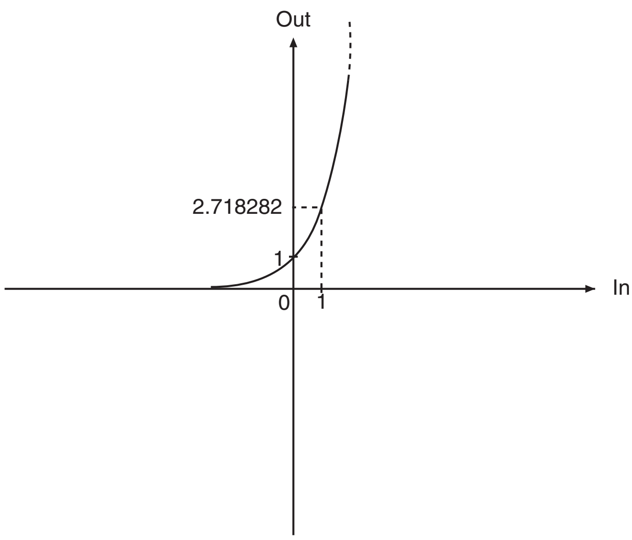
*1 If you omit an input parameter, the default value is not applied. A building error will occur.

*2 Negative numbers are excluded.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK	OK					
Out														OK	OK					

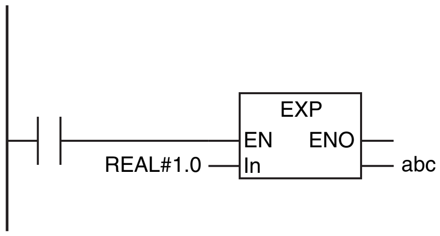
Function

The EXP instruction returns the value of e^{In}, where e is Euler's constant and *In* is an input variable.



The following example is for when *In* is REAL#1.0. The value of variable *abc* will be REAL#2.718282.

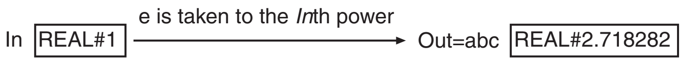
LD



ST

```
abc:=EXP(REAL#1.0);
```

The EXP instruction returns the natural logarithm of *In* to the base e.
The value of e^1 is 2.718282, so the value of *abc* will be REAL#2.718282.



Additional Information

- Use the EXPT (**) instruction (page 2-211) to find the powers of numbers with bases other than e.
- Use the CheckReal instruction (page 2-237) to see if *Out* is positive infinity, negative infinity, or nonnumeric data.

Precautions for Correct Use

- If the value of *In* is 0.0, positive infinity, negative infinity, or nonnumeric data, the value of *Out* is as shown below.

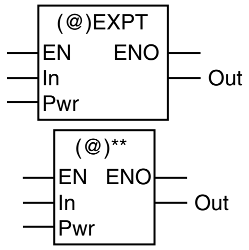
Value of *In*	Value of *Out*	
	Other than the right	NX1P2
0	1.0	1.0
+∞	Nonnumeric data	+∞
−∞	Nonnumeric data	0.0
Nonnumeric data	Nonnumeric data	Nonnumeric data

- If you pass an integer parameter to *In*, the data type is converted as follows:

Data type of parameter that is passed to *In*	Data type of *In*
USINT, UINT, SINT, or INT	REAL
UDINT or DINT	LREAL
ULINT or LINT	A building error will occur.

EXPT (**)

The EXPT (**) instruction raises one real number to the power of another real number.

Instruction	Name	FB/FUN	Graphic expression	ST expression
EXPT (**)	Exponentiation	FUN		Out:=EXPT(In, Pwr); Out:=In ** Pwr;

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Base number	Input	Base number (e.g., 5 for 5^2)	Depends on data type.	---	*
Pwr	Exponent		Exponent (e.g., 2 for 5^2)			
Out	Calculation result	Output	Calculation result	Depends on data type.	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

The Instruction of LD and the EXPT Instruction in ST

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK	OK					
Pwr														OK	OK					
Out														OK	OK					

The ** Operator in ST

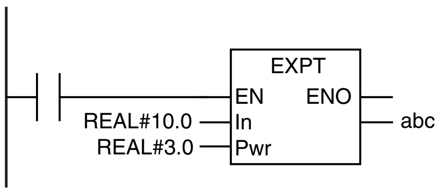
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Pwr						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Out						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					

Function

The EXPT (**) instruction raises base number *In* to exponent *Pwr* to find In^{Pwr}.

The following example is for when *In* is REAL#10.0 and *Pwr* is REAL#3.0. The value of variable *abc* will be REAL#1000.0.

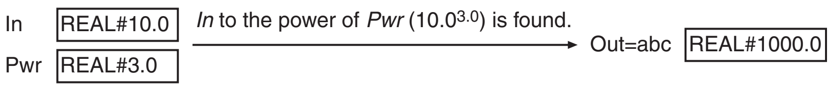
LD



ST

```
abc:=EXPT(REAL#10.0, REAL#3.0);
```

The ACOS instruction finds *In* to the power of *Pwr*.
$10.0^{3.0}$ is 1,000.0, so the value of *abc* will be REAL#1000.0.



Difference in Specifications between Ladder Diagrams and ST

Specifications of this instruction depend on whether it is used in a ladder diagram or the ** operator is used in ST. The following table gives the differences in specifications. The specifications of the EXPT instruction and the ** instruction of ladder diagrams and the EXPT function of ST are exactly the same.

Item	EXPT functions in ladder diagrams and ST	** operator in ST
Existence of the *EN* and *EN0* variables	Present	None
Number of data processing bits if *In* and *Pwr* are integer data	32 or 64*1	64*2

*1. Operation is performed with the smaller real number included, from either REAL or LREAL data. For example, if you operate SINT and DINT data, the data processing bits will be aligned to the size of LREAL data, i.e., 64-bit processing is performed.

*2. 64-bit processing is performed. For example, if two SINT values are exponentiated, 64-bit processing is performed.

Additional Information

- Use the EXP instruction (page 2-209) to find powers of base e.
- Use the CheckReal instruction (page 2-237) to see if *Out* is positive infinity, negative infinity, or non-numeric data.

Precautions for Correct Use

- If the absolute value of the calculation result is lower than the minimum value for a real number, the value of *Out* will be 0.

  Example: $(1.175494e{-}38)^2 \rightarrow 0$
- An error will not occur even if an underflow or overflow occurs in the calculation when the ** operator is used.
- If an underflow or overflow occurs in the calculation when the ** operator is used, the calculation result may not be as expected. Allow sufficient leeway in the sizes of the data types for input and output parameters so that overflows and underflows do not occur.
- For the EXPT instruction and ** instruction of ladder diagrams and the EXPT function of ST, if you pass an integer parameter to *In*, the data type is converted as follows:

Data type of parameter that is passed to *In*	Data type of *In*
USINT, UINT, SINT, or INT	REAL
UDINT or DINT	LREAL
ULINT or LINT	A building error will occur.

- When the ** operator is used, if you select the version 1.15 or earlier in the Select Device Area of the Project Properties Dialog Box on the Sysmac Studio for an NX701 CPU Unit, NJ-series CPU Unit and NY-series Controller, integer variables are calculated as real number variables even if they set as operands.
  If a rounding error is included in the result of calculations, the result may not be an intended value because all values after the decimal point are truncated.
  Use the EXPT and TO_** (Integer Conversion Group) instructions together to round values after the decimal point.
  Example) TO_INT (EXPT(X,Y))

Combination of *In* and *Pwr* values

The following table shows the values of *Out* for different combinations of *In* and *Pwr* values.

● The EXPT Function for a Device Other Than the NX1P2 CPU Unit

<table>
<tr><th colspan="2" rowspan="2"></th><th colspan="10">In</th></tr>
<tr><th>+∞</th><th>1 to +∞</th><th>1</th><th>0 to 1</th><th>0</th><th>-1 to 0</th><th>-1</th><th>-1 to -∞</th><th>-∞</th><th>Non-numeric data</th></tr>
<tr><th rowspan="11">Pwr</th><th>+∞</th><td>+∞</td><td>+∞</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>+∞</td><td>1</td><td>Non-numeric data</td></tr>
<tr><th>Positive even number</th><td rowspan="3">+∞</td><td rowspan="3">Number *1*2</td><td rowspan="3">1</td><td rowspan="3">Number*1*2</td><td rowspan="3">0</td><td>Number*1*2</td><td>1</td><td>Number*1*2</td><td rowspan="3">+∞</td><td rowspan="3">Non-numeric data</td></tr>
<tr><th>Positive odd number</th><td>Number*2*3</td><td>-1</td><td>Number*2*3</td></tr>
<tr><th>Positive decimal number</th><td colspan="3">Nonnumeric data</td></tr>
<tr><th>0</th><td>1</td><td colspan="3">1</td><td>1</td><td colspan="3">1</td><td>1</td><td>1</td></tr>
<tr><th>Negative even number</th><td rowspan="3">0</td><td rowspan="3">Number*1*2</td><td rowspan="3">1</td><td rowspan="3">Number*1*2</td><td rowspan="3">+∞</td><td>Number*1*2</td><td>1</td><td>Number*1*2</td><td rowspan="3">0</td><td rowspan="3">Non-numeric data</td></tr>
<tr><th>Negative odd number</th><td>Number*2*3</td><td>-1</td><td>Number*2*3</td></tr>
<tr><th>Negative decimal number</th><td colspan="3">Nonnumeric data</td></tr>
<tr><th>-∞</th><td>0</td><td>0</td><td>1</td><td>+∞</td><td>+∞</td><td>+∞</td><td>1</td><td>0</td><td>0</td><td>Non-numeric data</td></tr>
<tr><th>Nonnumeric data</th><td>1</td><td>Non-numeric data</td><td>1</td><td>Non-numeric data</td><td>1</td><td colspan="3">Nonnumeric data</td><td>1</td><td>Non-numeric data 1</td></tr>
</table>

*1 If the calculation result exceeds the valid range of the data type of *Out*, the value of *Out* will be positive infinity.

*2 If the calculation result is too close to 0 to express with the data type of *Out* or if it is an unnormalized number, the value of *Out* will be 0.

*3 If the calculation result exceeds the valid range of the data type of *Out*, the value of *Out* will be negative infinity.

● The EXPT Function for the NX1P2 CPU Unit

<table>
<tr><th colspan="2" rowspan="2"></th><th colspan="10">In</th></tr>
<tr><th>+∞</th><th>1 to +∞</th><th>1</th><th>0 to 1</th><th>0</th><th>-1 to 0</th><th>-1</th><th>-1 to -∞</th><th>-∞</th><th>Non-numeric data</th></tr>
<tr><th rowspan="10">Pwr</th><th>+∞</th><td>+∞</td><td>+∞</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>+∞</td><td>1</td><td>Non-numeric data</td></tr>
<tr><th>Positive even number</th><td rowspan="3">+∞</td><td rowspan="3">Number *1*2</td><td rowspan="3">1</td><td rowspan="3">Number*1*2</td><td rowspan="3">0</td><td>Number*1*2</td><td>1</td><td>Number*1*2</td><td>+∞</td><td rowspan="3">Non-numeric data</td></tr>
<tr><th>Positive odd number</th><td>Number*2*3</td><td>-1</td><td>Number*2*3</td><td>-∞</td></tr>
<tr><th>Positive decimal number</th><td colspan="3">Nonnumeric data</td><td>+∞</td></tr>
<tr><th>0</th><td>1</td><td colspan="3">1</td><td>1</td><td colspan="3">1</td><td>1</td><td>1</td></tr>
<tr><th>Negative even number</th><td rowspan="3">0</td><td rowspan="3">Number*1*2</td><td rowspan="3">1</td><td rowspan="3">Number*1*2</td><td rowspan="3">+∞</td><td>Number*1*2</td><td>1</td><td>Number*1*2</td><td>0</td><td rowspan="3">Non-numeric data</td></tr>
<tr><th>Negative odd number</th><td>Number*2*3</td><td>-1</td><td>Number*2*3</td><td>-0</td></tr>
<tr><th>Negative decimal number</th><td colspan="3">Nonnumeric data</td><td>0</td></tr>
<tr><th>-∞</th><td>0</td><td>0</td><td>1</td><td>+∞</td><td>+∞</td><td>+∞</td><td>1</td><td>0</td><td>0</td><td>Non-numeric data</td></tr>
<tr><th>Nonnumeric data</th><td>1</td><td>Non-numeric data</td><td>1</td><td>Non-numeric data</td><td>1</td><td colspan="3">Nonnumeric data</td><td>1</td><td>Non-numeric data
1</td></tr>
</table>

*1 If the calculation result exceeds the valid range of the data type of *Out*, the value of *Out* will be positive infinity.

*2 If the calculation result is too close to 0 to express with the data type of *Out* or if it is an unnormalized number, the value of *Out* will be 0.

*3 If the calculation result exceeds the valid range of the data type of *Out*, the value of *Out* will be negative infinity.

● The ** Operator

<table>
<tr><th colspan="2" rowspan="2"></th><th colspan="10">In</th></tr>
<tr><th>+∞</th><th>1 to +∞</th><th>1</th><th>0 to 1</th><th>0</th><th>-1 to 0</th><th>-1</th><th>-1 to -∞</th><th>-∞</th><th>Non-numeric data</th></tr>
<tr><th rowspan="10">Pwr</th><th>+∞</th><td>+∞</td><td>+∞</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>+∞</td><td>1</td><td>Nonnumeric data</td></tr>
<tr><th>Positive even number</th><td rowspan="3">+∞</td><td rowspan="3">Number *1*2*3*</td><td rowspan="3">1</td><td rowspan="3">Number *1*2</td><td rowspan="3">0</td><td>Number *1*2</td><td>1</td><td>Number *1*2*3</td><td>+∞</td><td rowspan="3">Nonnumeric data</td></tr>
<tr><th>Positive odd number</th><td>Number *2*4</td><td>-1</td><td>Number *2*3*4</td><td>-∞</td></tr>
<tr><th>Positive decimal number</th><td colspan="3">Nonnumeric data</td><td>+∞</td></tr>
<tr><th>0</th><td>1</td><td colspan="3">1</td><td>1</td><td colspan="3">1</td><td>1</td><td>1</td></tr>
<tr><th>Negative even number</th><td rowspan="3">0</td><td rowspan="3">Number*1*2</td><td rowspan="3">1</td><td rowspan="3">Number*1*2</td><td rowspan="3">+∞*5</td><td>Number*1*2</td><td>1</td><td>Number*1*2</td><td>0</td><td rowspan="3">Nonnumeric data</td></tr>
<tr><th>Negative odd number</th><td>Number*2*4</td><td>-1</td><td>Number*2*4</td><td>-0</td></tr>
<tr><th>Negative decimal number</th><td colspan="3">Nonnumeric data</td><td>0</td></tr>
<tr><th>-∞</th><td>0</td><td>0</td><td>1</td><td>+∞</td><td>+∞</td><td>+∞</td><td>1</td><td>0</td><td>0</td><td>Nonnumeric data</td></tr>
<tr><th>Nonnumeric data</th><td>1</td><td colspan="3">Nonnumeric data</td><td>1</td><td colspan="3">Nonnumeric data</td><td>1</td><td>Nonnumeric data 1</td></tr>
</table>

*1 If the calculation result exceeds the valid range of the data type of *Out*, the value of *Out* will be positive infinity.

*2 If the calculation result is too close to 0 to express with the data type of *Out* or if it is an unnormalized number, the value of *Out* will be 0.

*3 When both *In* and *Pwr* are integer data, if the calculation result exceeds the valid range of the data type of *Out*, *Out* will contain an undefined value.

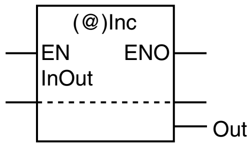
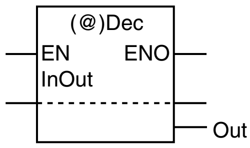
*4 If the calculation result exceeds the valid range of the data type of *Out*, the value of *Out* will be negative infinity.

*5 When both *In* and *Pwr* are integer data, Out will contain an undefined value.

Inc and Dec

Inc: Increments an integer value.

Dec: Decrements an integer value.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Inc	Increment	FUN		Inc(InOut);
Dec	Decrement	FUN		Dec(InOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
InOut	Target data	In-out	Target data	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
InOut						OK	OK	OK	OK	OK	OK	OK	OK							
Out	OK																			

Function

● Inc

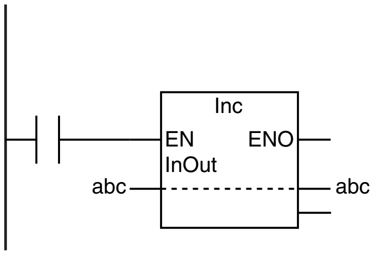
The Inc instruction increments target data *InOut*. If the result exceeds the maximum value of *InOut*, *InOut* returns to the minimum value.

● Dec

The Dec instruction decrements target data *InOut*. If the result exceeds the minimum value of *InOut*, *InOut* returns to the maximum value.

The following example for the Inc instruction is for when variable *abc* is passed to *InOut*. If the value of *abc* is INT#4, the value of *abc* after the instruction is executed will be INT#5.

LD



ST

```
Inc(abc);
```

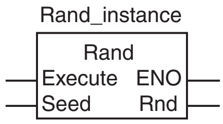
The Inc instruction increments *InOut*.
If the value of *abc* is INT#4, the value of *abc* after the instruction is executed will be INT#5.

Precautions for Correct Use

Return value *Out* is not used when the instruction is used in ST.

Rand

The Rand instruction generates pseudorandom numbers.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Rand	Random Number	FB		Rand_instance(Execute, Seed, Rnd);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Seed	Random number pattern	Input	Random number pattern 0: Not specified	Depends on data type.	---	*1
Rnd	Random number	Output	Random number	*2	---	---

*1 If you omit the input parameter, the value will be 0. It will not be the value that is specified for the Initial Value attribute.

*2 0.00000000000000e+0 to 1.00000000000000e+0

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Seed							OK													
Rnd															OK					

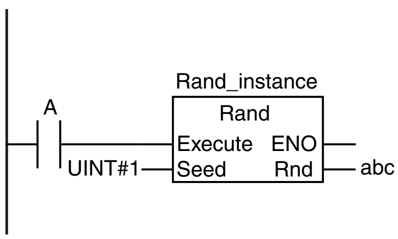
Function

The Rand instruction specifies random number *Rnd*. The value of *Rnd* is different each time the instruction is executed: Random number pattern *Seed* specifies the random number system. If the value of *Seed* is the same, the same random number series is generated each time the power supply is turned ON. This allows you to generate a repeatable series of random numbers.

If the value of *Seed* is 0, random numbers that cannot be repeated are generated. If you do not want to generate the same series of random numbers each time the power supply is turned ON, set the value of *Seed* to 0.

The following programming example is for when *Seed* is UINT#1. The value of *Seed* is not 0, so random numbers that can be repeated are generated.

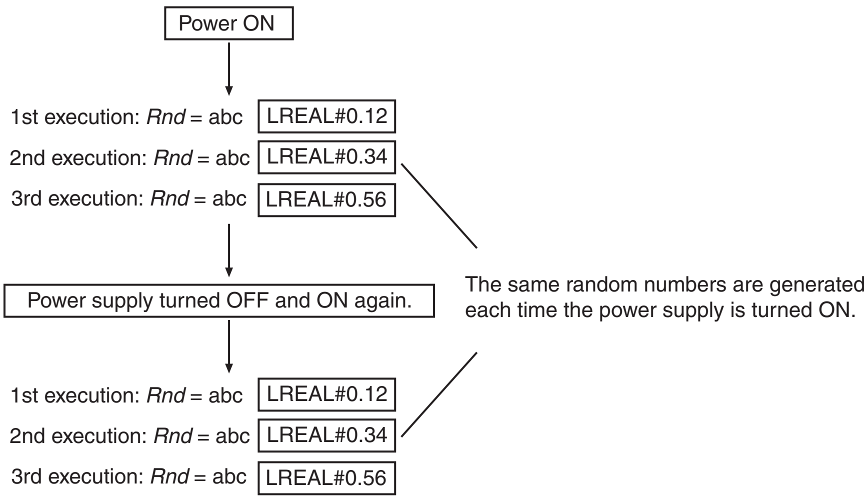
LD



ST

```
Rand_instance(A, UINT#1, abc);
```

The Rand instruction generates a repeatable series of random numbers.



* The values of the random numbers that are given above are examples. The actual values will be different.

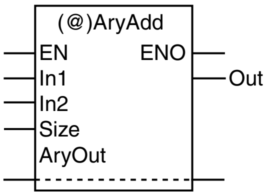
Additional Information

The value of *Rnd* is a real number between 0 and 1. Use the following processing to generate random numbers within a specific range.

Example: The following formula generates random numbers between 100 and 200.
Rand_instance(A, UINT#1, abc);
Random number:=LREAL_TO_INT((200.0-100.0)*abc)+100;

AryAdd

The AryAdd instruction adds corresponding elements of two arrays.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryAdd	Array Addition	FUN		AryAdd(In1, In2, Size, Ary-Out);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1[] (array) and In2[] (array)	Array to process	Input	Array to process	Depends on data type.	---	*
Size	Number of elements to process	Input	Number of elements to process	Depends on data type.	---	1
AryOut[] (array)	Calculation results array	In-out	Calculation results array	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

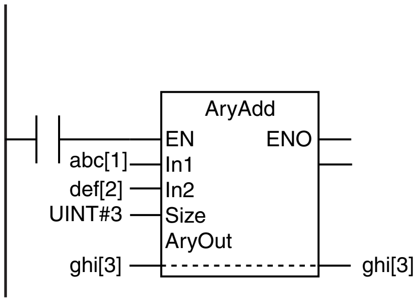
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[] (array)						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
In2[] (array)	Must be an array with the same data type as *In1[]*.																			
Size							OK													
AryOut[] (array)	Must be an array with the same data type as *In1[]*.																			
Out	OK																			

Function

The AryAdd instruction adds *Size* elements of arrays to process *In1[]* and *In2[]* starting from *In1[0]* and *In2[0]*. The results are assigned to corresponding elements of calculation results array *AryOut[]*.

The following example is for when *Size* is UINT#3.

LD



ST

```
AryAdd(abc[1], def[2], UINT#3, ghi[3]);
```

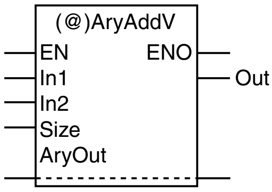
Size=UINT#3	In1[0]=abc[1]	1234	+	In2[0]=def[2]	2345	→	AryOut[0]=ghi[3]	3579
	In1[1]=abc[2]	2345	+	In2[1]=def[3]	3456	→	AryOut[1]=ghi[4]	5801
	In1[2]=abc[3]	3456	+	In2[2]=def[4]	4567	→	AryOut[2]=ghi[5]	8023

Precautions for Correct Use

- Use the same data type for *In1[]*, *In2[]*, and *AryOut[]*. If they are different, a building error will occur.
- If the calculation results exceed the valid range of *AryOut[]*, the results will be illegal values. An error will not occur. Corruption will not occur in the data in the memory area adjacent to those elements.
- The values in *AryOut[]* do not change if the value of *Size* is 0.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *AryOut[]* will not change.
  - The value of *Size* exceeds the array range of *In1[], In2p[],* or *AryOut[].*

AryAddV

The AryAddV instruction adds the same value to specified elements of an array.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryAddV	Array Value Addition	FUN		AryAddV(In1, In2, Size, AryOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1[] (array)	Addition array	Input	Addition array	Depends on data type.	---	*
In2	Value to add		Value to add			
Size	Number of elements		Number of elements of *In1[]* for addition			1
AryOut[] (array)	Addition results array	In-out	Addition results array	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

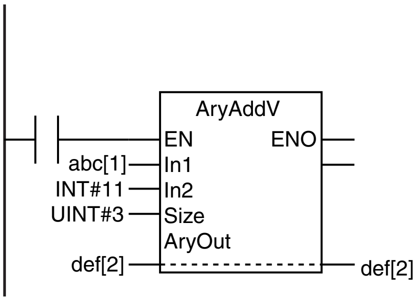
	Boolean	Bit string				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[] (array)						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
In2	Must be same data type as *In1[]*.																			
Size							OK													
AryOut[] (array)	Must be same data type as *In1[]*.																			
Out	OK																			

Function

The AryAddV instruction adds value to add *In2* to *Size* elements of addition array *In1[]* starting from *In1[0]*. It outputs the results to addition results array *AryOut[]*.

The following example is for when *In2* is INT#11 and *Size* is UINT#3.

LD



ST

```
AryAddV(abc[1], INT#11, UINT#3, def[2]);
```

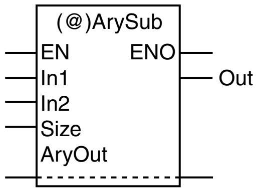
Size=UINT#3	In1[0]=abc[1]	12	+	In2=INT#11	→	AryOut[0]=def[2]	23
	In1[1]=abc[2]	23	+	In2=INT#11	→	AryOut[1]=def[3]	34
	In1[2]=abc[3]	34	+	In2=INT#11	→	AryOut[2]=def[4]	45

Precautions for Correct Use

- Use the same data type for *In1[]*, *In2*, and *AryOut[]*. If they are different, a building error will occur.
- If the addition results exceed the valid range of *AryOut[]*, the elements of *AryOut[]* will contain illegal values. An error will not occur. Corruption will not occur in the data in the memory area adjacent to those elements.
- The values in *AryOut[]* do not change if the value of *Size* is 0.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *AryOut[]* will not change.
  - If the value of *Size* exceeds the array area of *In1[]* or *AryOut[]*.

ArySub

The ArySub instruction subtracts corresponding elements of two arrays.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ArySub	Array Subtraction	FUN		ArySub(In1, In2, Size, Ary-Out);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1[] (array)	Minuend array	Input	Minuend array	Depends on data type.	---	*
In2[] (array)	Subtrahend array		Subtrahend array			
Size	Number of elements		Number of elements for sub-traction			1
AryOut[] (array)	Subtraction results array	In-out	Subtraction results array	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

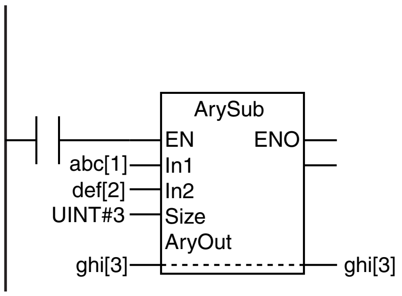
	Boolean	Bit string				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[] (array)						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
In2[] (array)	Must be same data type as *In1[]*.																			
Size							OK													
AryOut[] (array)	Must be same data type as *In1[]*.																			
Out	OK																			

Function

The ArySub instruction subtracts Size elements of subtrahend array *In2[]* from corresponding elements of minuend array *In1[]* starting with *In1[0]* and *In2[0]*. It outputs the subtraction results to subtraction results array *AryOut[]*.

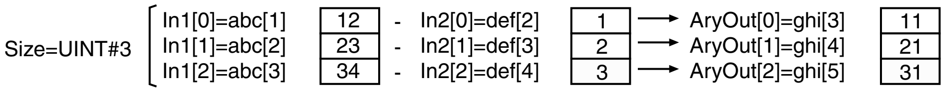
The following example is for when *Size* is UINT#3.

LD



ST

```
ArySub(abc[1], def[2], UINT#3, ghi[3]);
```

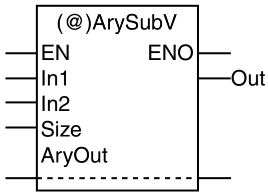


Precautions for Correct Use

- Use the same data type for *In1[]*, *In2[]*, and *AryOut[]*. If they are different, a building error will occur.
- If the subtraction results exceed the valid range of *AryOut[]*, the elements of *AryOut[]* will contain illegal values. An error will not occur. Corruption will not occur in the data in the memory area adjacent to those elements.
- The values in *AryOut[]* do not change if the value of *Size* is 0.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *AryOut[]* will not change.
    - The value of *Size* exceeds the array range of *In1[], In2[],* or *AryOut[].*

ArySubV

The ArySubV instruction subtracts the same value from specified elements of an array.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ArySubV	Array Value Subtraction	FUN		ArySubV(In1, In2, Size, AryOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1[] (array)	Minuend array	Input	Minuend array	Depends on data type.	---	*
In2	Subtrahend		Subtrahend			
Size	Number of elements		Number of elements of *In1[]* for subtraction			1
AryOut[] (array)	Subtraction results array	In-out	Subtraction results array	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

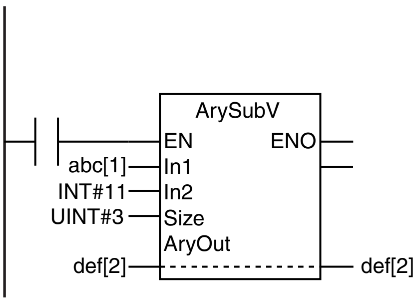
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[] (array)						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
In2	Must be same data type as the elements of *In1[]*.																			
Size							OK													
AryOut[] (array)	Must be same data type as *In1[]*.																			
Out	OK																			

Function

The ArySubV instruction subtracts subtrahend *In2* from *Size* elements of minuend array *In1[]* starting from *In1[0].* It outputs the results to subtraction results array *AryOut[]*.

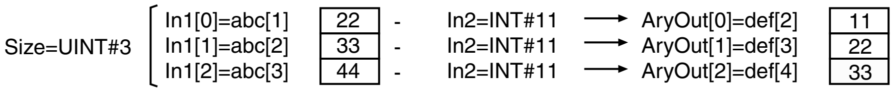
The following example is for when *In2* is INT#11 and *Size* is UINT#3.

LD



ST

```
ArySubV(abc[1], INT#11, UINT#3, def[2]);
```

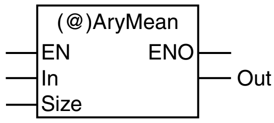


Precautions for Correct Use

- Use the same data type for *In1[]*, *In2*, and *AryOut[]*. If they are different, a building error will occur.
- If the subtraction results exceed the valid range of *AryOut[]*, the elements of *AryOut[]* will contain illegal values. An error will not occur. Corruption will not occur in the data in the memory area adjacent to those elements.
- The values in *AryOut[]* do not change if the value of *Size* is 0.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *AryOut[]* will not change.
  - The value of *Size* exceeds the array area of *In1[]* or *AryOut[]*.

AryMean

The AryMean instruction calculates the average of the elements of an array.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryMean	Array Mean	FUN		Out := AryMean(In, Size);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array to process	Input	Array to process	Depends on data type.	---	*
Size	Number of elements to process		Number of *In[]* elements			1
Out	Calculation result	Output	Calculation result	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

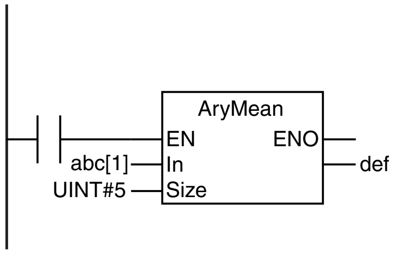
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK				
Size							OK													
Out						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK				

Function

The AryMean instruction calculates the average of *Size* elements of array to process *In[]* starting from *In[0]*.

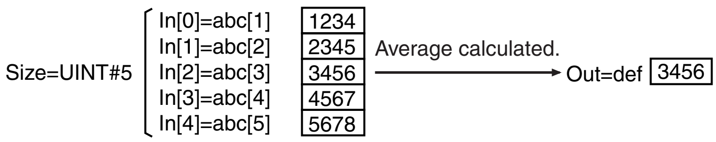
The following example is for when *Size* is UINT#5.

LD



ST

def:=AryMean(abc[1], UINT#5);

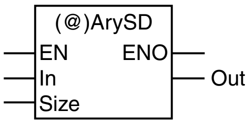


Precautions for Correct Use

- Refer to the descriptions of the functions of the ADD (+) instruction (page 2-166), SUB (–) instruction (page 2-174), MUL (*) instruction (page 2-181), and DIV (/) instruction (page 2-189) for the calculation results when the value of *In[]* is positive infinity, negative infinity, or nonnumeric data.
- If *In[]* or *Out* is an integer, the decimal portion of the average is truncated.
- If you use a different data type for *In[]* and *Out*, make sure the valid range of *Out* includes the valid range of *In[]*.
- If the calculation result exceeds the valid range of *Out*, *Out* will contain an illegal value. An error will not occur.
- If an intermediate value in the calculation process exceeds the valid range of *IN[]*, *Out* will contain an illegal value. An error will not occur.
- If the value of *Size* is 0, the value of *Out* is 0.
- An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.
  - The value of *Size* exceeds the array area of *In[]*.

ArySD

The ArySD instruction calculates standard deviation of the elements of an array.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ArySD	Array Element Standard Deviation	FUN		Out:=ArySD(In, Size);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array to process	Input	Array to process	Depends on data type.	---	*
Size	Number of elements		Number of elements of *In[]* for conversion			2
Out	Standard deviation	Output	Standard deviation	Depends on data type.	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)														OK	OK					
Size							OK													
Out														OK	OK					

Function

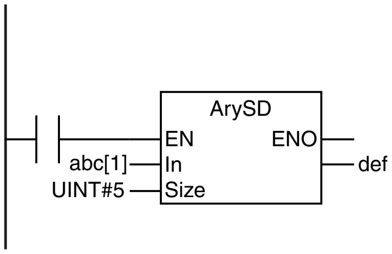
The ArySD instruction calculates the standard deviation of *Size* elements of array to process *In[]* starting from *In[0]*.

$$\text{Standard deviation} = \sqrt{\frac{\sum_{i}(In[i]-InM)^{2}}{Size-1}}$$

i: Subscript of *In[]*, 0 to *Size* – 1
InM: Average value of *In[0]* to *In[Size - 1]*

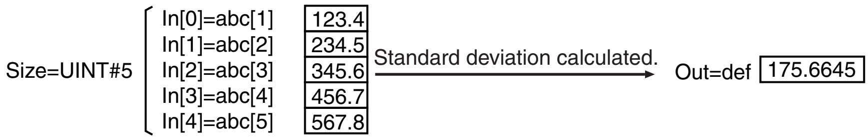
The following example is for when *Size* is UINT#5.

LD



ST

```
def:=ArySD(abc[1], UINT#5);
```

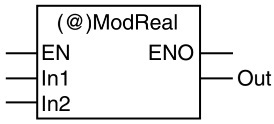


Precautions for Correct Use

- If the value of *Size* is 0 or 1, the value of *Out* is 0.
- If an intermediate value in the calculation process exceeds the valid range of *IN[]*, *Out* will contain an illegal value. An error will not occur.
- An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.
    - The value of *Size* exceeds the array area of *In[]*.

ModReal

The ModReal instruction calculates the remainder of real number division.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ModReal	Real Number Modulo-division	FUN		Out:=ModReal(In1, In2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Dividend	Input	Dividend	Depends on data type.	---	*
In2	Divisor		Divisor			
Out	Remainder	Output	Remainder	Depends on data type.	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

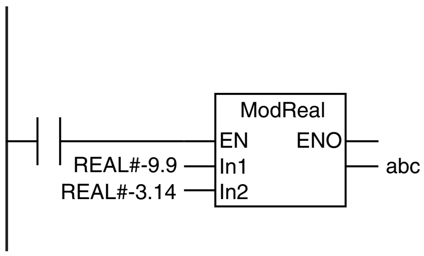
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1														OK	OK					
In2														OK	OK					
Out														OK	OK					

Function

The ModReal instruction divides dividend *In1* by divisor *In2* to find the remainder.

The following example is for when *In1* is REAL#–9.9 and *In2* is REAL#–3.14. The value of variable *abc* will be REAL#–0.48.

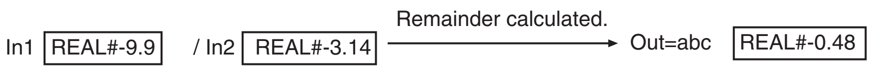
LD



ST

abc:=ModReal(REAL#-9.9, REAL#-3.14);

The ModReal instruction divides *In1* by *In2* to find the remainder.
The remainder of –9.9/(–3.14) is –0.48, so the value of *abc* will be REAL#–0.48.



Additional Information

Use the CheckReal instruction (page 2-237) to see if the value of *Out* is positive infinity, negative infinity, or nonnumeric data.

Precautions for Correct Use

- The following table shows the values of *Out* for different combinations of *In1* and *In2* values.

		In1				
		0	**Number**	**+∞**	**−∞**	**Nonnumeric data**
In2	**0**	Nonnumeric data	Nonnumeric data	Nonnumeric data	Nonnumeric data	Nonnumeric data
	Number	0	Remainder of *In1/In2*	Nonnumeric data	Nonnumeric data	Nonnumeric data
	+∞	0	Value of *In1*	Nonnumeric data	Nonnumeric data	Nonnumeric data
	−∞	0	Value of *In1*	Nonnumeric data	Nonnumeric data	Nonnumeric data
	Nonnumeric data	Nonnumeric data	Nonnumeric data	Nonnumeric data	Nonnumeric data	Nonnumeric data

- If you pass an integer parameter to *In1* or *In2*, the data type is converted as follows:

Data type of parameter that is passed to *In1* or *In2*	Data type of *In1* or *In2*
USINT, UINT, SINT, or INT	REAL
UDINT or DINT	LREAL
ULINT or LINT	A building error will occur.

Fraction

The Fraction instruction finds the fractional part of a real number.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Fraction	Real Number Fraction	FUN	(@)Fraction EN ENO In Out	Out:=Fraction(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Real number	Input	Real number	Depends on data type.	---	*
Out	Fractional part	Output	Fractional part	Depends on data type.	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

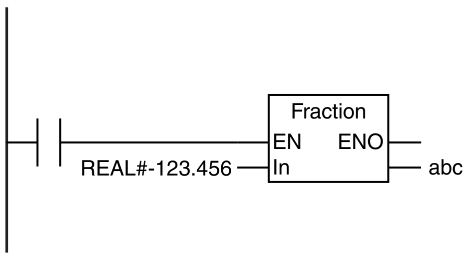
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK	OK					
Out														OK	OK					

Function

The Fraction instruction finds the fractional part of real number *In*.

The following example is for when *In* is REAL#–123.456. The value of variable *abc* will be REAL#–0.456.

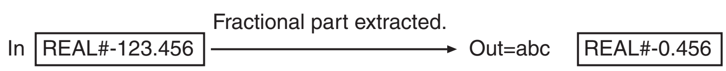
LD



ST

abc:=Fraction(REAL#-123.456);

The Fraction instruction finds the fractional part of *In*.
The fractional part of –123.456 is –0.456, so the value of *abc* will be REAL#0.456.



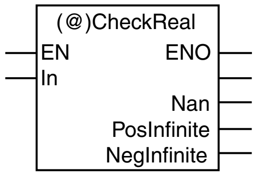
Additional Information

- Use the CheckReal instruction (page 2-237) to see if the value of *Out* is positive infinity, negative infinity, or nonnumeric data.
- If you pass an integer parameter to *In*, the data type is converted as follows:

Data type of parameter that is passed to *In*	Data type of *In*
USINT, UINT, SINT, or INT	REAL
UDINT or DINT	LREAL
ULINT or LINT	A building error will occur.

CheckReal

The CheckReal instruction checks a real number to see if it is infinity or nonnumeric data.

Instruction	Name	FB/FUN	Graphic expression	ST expression
CheckReal	Real Number Check	FUN		CheckReal(In, Nan, PosIn-finite, NegInfinite);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Real number	Input	Real number	Depends on data type.	---	*
Out	Return value	Output	Always TRUE	TRUE only	---	---
Nan	Nonnumeric data check result		TRUE: Nonnumeric data FALSE: Not nonnumeric data	Depends on data type.		
PosInfinite	Positive infinity check result		TRUE: Positive infinity FALSE: Not positive infinity			
NegInfinite	Negative infinity check result		TRUE: Negative infinity FALSE: Not negative infinity			

* If you omit an input parameter, the default value is not applied. A building error will occur.

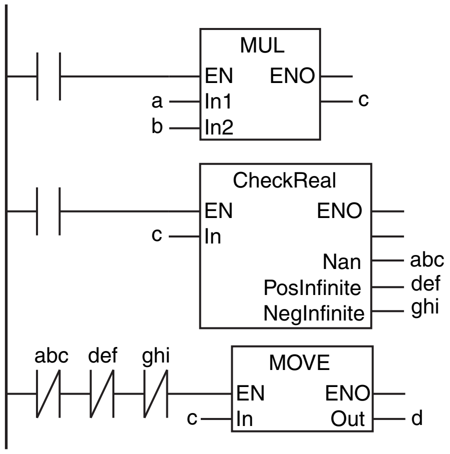
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK	OK					
Out	OK																			
Nan	OK																			
PosInfinite	OK																			
NegInfinite	OK																			

Function

The CheckReal instruction checks a real number In to see if it is nonnumeric data, positive infinity, or negative infinity. It outputs the results to *Nan*, *PosInfinite*, and *NegInfinite*.

The following figure shows a programming example. The values of REAL variables *a* and *b* are multiplied and the result is tested to see if it is a real number. If the multiplication result is a real number, it is assigned to variable *d*.

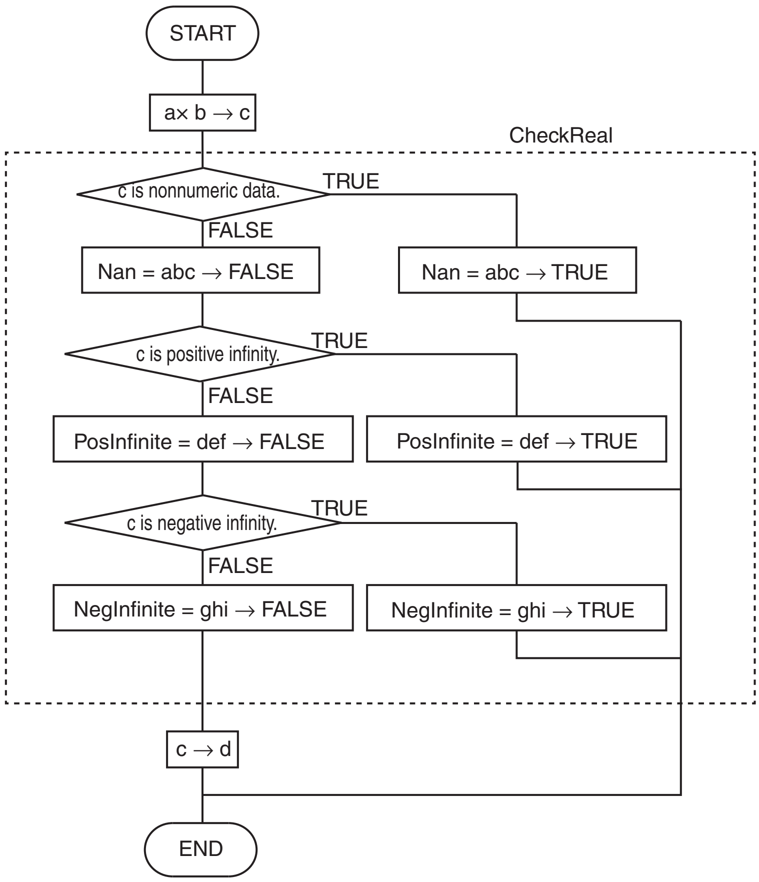
LD



ST

```
c:=a*b;
CheckReal(c, abc, def, ghi);
IF ( (abc=FALSE) AND (def=FALSE) AND (ghi=FALSE) ) THEN
    d:=c;
END_IF;
```

If the product *c* of *a* and *b* is not nonnumeric data, positive infinity, or negative infinity, then the value of *c* is assigned to *d*.



Additional Information

Use this instruction on the result of a math instruction that handles real numbers to see if the result is nonnumeric data, positive infinity, or negative infinity.

Precautions for Correct Use

- Return value *Out* is not used when the instruction is used in ST.
- If you pass an integer parameter to *In*, the data type is converted as follows:

Data type of parameter that is passed to *In*	Data type of *In*
USINT, UINT, SINT, or INT	REAL
UDINT or DINT	LREAL
ULINT or LINT	A building error will occur.

BCD Conversion Instructions

_BCD_TO_*

These instructions convert BCD bit strings into unsigned integers.

Instruction	Name	FB/FUN	Graphical expression	ST expression
_BCD_TO_*	BCD-to-Unsigned Integer Conversion Group	FUN	(@)**_BCD_TO_*** EN ENO In Out "**" must be a bit string data type. "***" must be an integer data type.	Out:=**_BCD_TO_*** (In); "**" must be a bit string data type. "***" must be an integer data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*	---	0
Out	Conversion result	Output	Conversion result	*	---	---

* The valid ranges depend on the data types of *In* and *Out*. Refer to *Function*, below, for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Out						OK	OK	OK	OK	OK	OK	OK	OK							

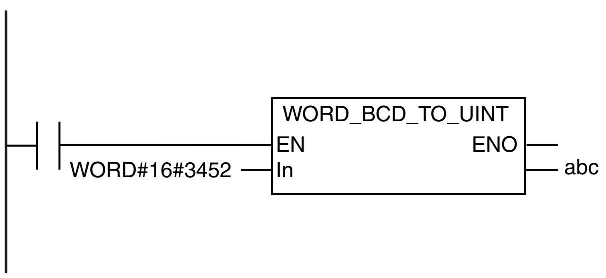
Function

These instructions convert data to convert *In* (which must be a BCD bit string) into an unsigned integer.

The name of the instruction is determined by the data types of *In* and conversion result *Out*. For example, if *In* is WORD data and *Out* is UINT data, the name of the instruction is WORD_BCD_TO_UINT.

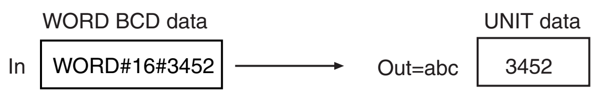
The following example for the WORD_BCD_TO_UINT instruction is for when *In* is WORD16#3452.

LD



ST

```
abc:=WORD_BCD_TO_UINT(WORD#16#3452);
```



The following table shows the valid ranges for *In* and *Out* according to their data types.

Data type of *In*	Data type of *Out*	Valid range for *In*	Valid range for *Out*
BYTE	USINT	16#00 to 16#99 (BCD)	0 to 99
	UINT		
	UDINT		
	ULINT		
	SINT		
	INT		
	DINT		
	LINT		
WORD	USINT	16#0000 to 16#0255 (BCD)	0 to 255
	UINT	16#0000 to 16#9999 (BCD)	0 to 9999
	UDINT		
	ULINT		
	SINT	16#0000 to 16#0127 (BCD)	0 to 127
	INT	16#0000 to 16#9999 (BCD)	0 to 9999
	DINT		
	LINT		
DWORD	USINT	16#0000_0000 to 16#0000_0255 (BCD)	0 to 255
	UINT	16#0000_0000 to 16#0006_5535 (BCD)	0 to 65535
	UDINT	16#0000_0000 to 16#9999_9999 (BCD)	0 to 99999999
	ULINT		
	SINT	16#0000_0000 to 16#0000_0127 (BCD)	0 to 127
	INT	16#0000_0000 to 16#0003_2767 (BCD)	0 to 32767
	DINT	16#0000_0000 to 16#9999_9999 (BCD)	0 to 99999999
	LINT		
LWORD	USINT	16#0000_0000_0000_0000 to 16#0000_0000_0000_0255 (BCD)	0 to 255
	UINT	16#0000_0000_0000_0000 to 16#0000_0000_0006_5535 (BCD)	0 to 65535
	UDINT	16#0000_0000_0000_0000 to 16#0000_0042_9496_7295 (BCD)	0 to 4294967295
	ULINT	16#0000_0000_0000_0000 to 16#9999_9999_9999_9999 (BCD)	0 to 9999999999999999
	SINT	16#0000_0000_0000_0000 to 16#0000_0000_0000_0127 (BCD)	0 to 127
	INT	16#0000_0000_0000_0000 to 16#0000_0000_0003_2767 (BCD)	0 to 32767
	DINT	16#0000_0000_0000_0000 to 16#0000_0021_4748_3647 (BCD)	0 to 2147483647
	LINT	16#0000_0000_0000_0000 to 16#9999_9999_9999_9999 (BCD)	0 to 9999999999999999

Additional Information

- To convert a BCD bit string to an integer, use a BCD_TO_** instruction (page 2-247).
- To convert an integer to a BCD bit string, use a **_TO_BCD_*** instruction (page 2-245).

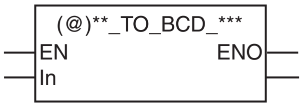
Precautions for Correct Use

- Always use the correct instruction name for the data types of *In* and *Out*.
- If the data size of *Out* is larger than the data size of *In*, the upper digits of *Out* will contain 0.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.

- The value of *In* is outside of the valid range.
- The value in *In* is not BCD bit string data (i.e., contains A, B, C, D, E, or F hexadecimal).

_TO_BCD_*

These instructions convert unsigned integers to BCD bit strings.

Instruction	Name	FB/FUN	Graphic expression	ST expression
_TO_BCD_*	Unsigned Integer-to-BCD Conversion Group	FUN	 "**" must be an integer data type. "***" must be a bit string data type.	Out:=**_TO_BCD_*** (In); "**" must be an integer data type. "***" must be a bit string data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*	---	0
Out	Conversion result	Output	Conversion result	*	---	---

* The valid ranges depend on the data types of *In* and *Out*. Refer to *Function*, below, for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						OK	OK	OK	OK	OK	OK	OK	OK							
Out		OK	OK	OK	OK															

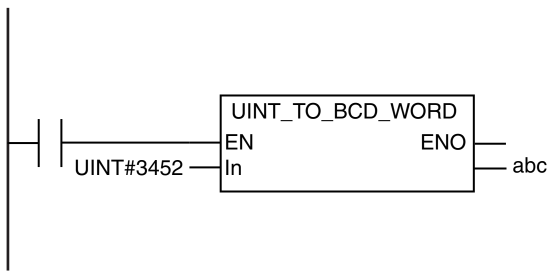
Function

These instructions convert data to convert *In* (which must be an unsigned integer) to a BCD bit string.

The name of the instruction is determined by the data types of *In* and conversion result *Out*. For example, if *In* is UINT data and *Out* is WORD data, the name of the instruction is UINT_TO_BCD_WORD.

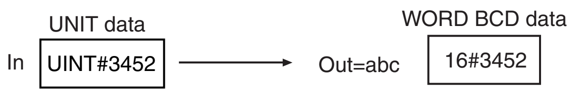
The following example for the UINT_TO_BCD_WORD instruction is for when *In* is UNIT#3452.

LD



ST

abc:=UINT_TO_BCD_WORD(UINT#3452);



The following table shows the valid ranges for *In* and *Out* according to their data types.

Data type of *In*	Data type of *Out*	Valid range for *In*	Valid range for *Out*
USINT	BYTE	0 to 99	16#00 to 16#99 (BCD)
	WORD	0 to 255	16#0000 to 16#0255 (BCD)
	DWORD		16#0000_0000 to 16#000_0255 (BCD)
	LWORD		16#0000_0000_0000_0000 to 16#0000_0000_0000_0255 (BCD)
UINT	BYTE	0 to 99	16#00 to 16#99 (BCD)
	WORD	0 to 9999	16#0000 to 16#9999 (BCD)
	DWORD	0 to 65535	16#0000_0000 to 16#0006_5535 (BCD)
	LWORD		16#0000_0000_0000_0000 to 16#0000_0000_0006_5535 (BCD)
UDINT	BYTE	0 to 99	16#00 to 16#99 (BCD)
	WORD	0 to 9999	16#0000 to 16#9999 (BCD)
	DWORD	0 to 99999999	16#0000_0000 to 16#9999_9999 (BCD)
	LWORD	0 to 4294967295	16#0000_0000_0000_0000 to 16#0000_0042_9496_7295 (BCD)
ULINT	BYTE	0 to 99	16#00 to 16#99 (BCD)
	WORD	0 to 9999	16#0000 to 16#9999 (BCD)
	DWORD	0 to 99999999	16#0000_0000 to 16#9999_9999 (BCD)
	LWORD	0 to 9999999999999999	16#0000_0000_0000_0000 to 16#9999_9999_9999_9999 (BCD)
SINT	BYTE	0 to 99	16#00 to 16#99 (BCD)
	WORD	0 to 127	16#0000 to 16#0127 (BCD)
	DWORD		16#0000_0000 to 16#0000_0127 (BCD)
	LWORD		16#0000_0000_0000_0000 to 16#0000_0000_0000_0127 (BCD)
INT	BYTE	0 to 99	16#00 to 16#99 (BCD)
	WORD	0 to 9999	16#0000 to 16#9999 (BCD)
	DWORD	0 to 32767	16#0000_0000 to 16#0003_2767 (BCD)
	LWORD		16#0000_0000_0000_0000 to 16#0000_0000_0003_2767 (BCD)
DINT	BYTE	0 to 99	16#00 to 16#99 (BCD)
	WORD	0 to 9999	16#0000 to 16#9999 (BCD)
	DWORD	0 to 99999999	16#0000_0000 to 16#9999_9999 (BCD)
	LWORD	0 to 2147483647	16#0000_0000_0000_0000 to 16#0000_0021_4748_3647 (BCD)
LINT	BYTE	0 to 99	16#00 to 16#99 (BCD)
	WORD	0 to 9999	16#0000 to 16#9999 (BCD)
	DWORD	0 to 99999999	16#0000_0000 to 16#9999_9999 (BCD)
	LWORD	0 to 9999999999999999	16#0000_0000_0000_0000 to 16#9999_9999_9999_9999 (BCD)

Additional Information

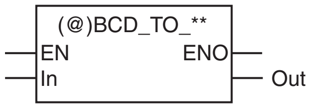
- To convert a specific BCD bit string to an integer, use a **_BCD_TO_*** instruction (page 2-242).
- To convert a BCD bit string to an integer, use a BCD_TO_** instruction (page 2-247).

Precautions for Correct Use

- Always use the correct instruction name for the data types of *In* and *Out*.
- If the data size of *Out* is larger than the data size of *In*, the upper digits of *Out* will contain 0.
- An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.
  - The value of *In* is outside of the valid range.

BCD_TO_**

The BCD_TO_** instruction converts BCD bit strings into unsigned integers.

Instruction	Name	FB/FUN	Graphic expression	ST expression
BCD_TO_**	BCD Data Type-to-Unsigned Integer Conversion Group	FUN	 "**" must be an integer data type.	Out:=BCD_TO_** (In); "**" must be an integer data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*1	---	*2
Out	Conversion result	Output	Conversion result	*1	---	---

*1 The valid ranges depend on the data types of *In* and *Out*. Refer to *Function*, below, for details.

*2 If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Out						OK	OK	OK	OK	OK	OK	OK	OK							

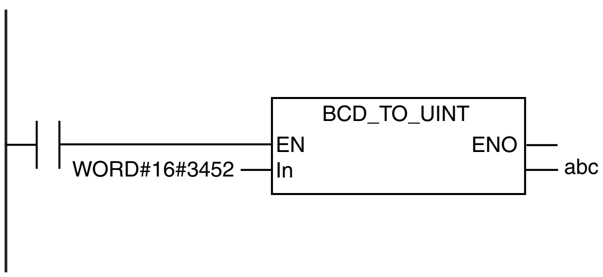
Function

These instructions convert data to convert *In* (which must be a BCD bit string) into an unsigned integer.

The name of the instruction is determined by the data type of conversion result *Out*. For example, if *Out* is the UINT data type, the instruction is BCD_TO_UINT.

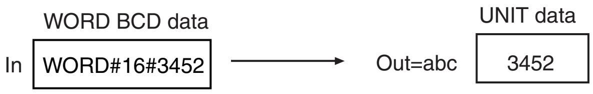
The following example for the BCD_TO_UINT instruction is for when *In* is WORD#16#3452.

LD



ST

abc:=BCD_TO_UINT(WORD#16#3452);



The following table shows the valid ranges for *In* and *Out* according to their data types.

Data type of *In*	Data type of *Out*	Valid range for *In*	Valid range for *Out*
BYTE	USINT	16#00 to 16#99 (BCD)	0 to 99
	UINT		
	UDINT		
	ULINT		
	SINT		
	INT		
	DINT		
	LINT		
WORD	USINT	16#0000 to 16#0255 (BCD)	0 to 255
	UINT	16#0000 to 16#9999 (BCD)	0 to 9999
	UDINT		
	ULINT		
	SINT	16#0000 to 16#0127 (BCD)	0 to 127
	INT	16#0000 to 16#9999 (BCD)	0 to 9999
	DINT		
	LINT		
DWORD	USINT	16#0000_0000 to 16#0000_0255 (BCD)	0 to 255
	UINT	16#0000_0000 to 16#0006_5535 (BCD)	0 to 65535
	UDINT	16#0000_0000 to 16#9999_9999 (BCD)	0 to 99999999
	ULINT		
	SINT	16#0000_0000 to 16#0000_0127 (BCD)	0 to 127
	INT	16#0000_0000 to 16#0003_2767 (BCD)	0 to 32767
	DINT	16#0000_0000 to 16#9999_9999 (BCD)	0 to 99999999
	LINT		
LWORD	USINT	16#0000_0000_0000_0000 to 16#0000_0000_0000_0255 (BCD)	0 to 255
	UINT	16#0000_0000_0000_0000 to 16#0000_0000_0006_5535 (BCD)	0 to 65535
	UDINT	16#0000_0000_0000_0000 to 16#0000_0042_9496_7295 (BCD)	0 to 4294967295
	ULINT	16#0000_0000_0000_0000 to 16#9999_9999_9999_9999 (BCD)	0 to 9999999999999999
	SINT	16#0000_0000_0000_0000 to 16#0000_0000_0000_0127 (BCD)	0 to 127
	INT	16#0000_0000_0000_0000 to 16#0000_0000_0003_2767 (BCD)	0 to 32767
	DINT	16#0000_0000_0000_0000 to 16#0000_0021_4748_3647 (BCD)	0 to 2147483647
	LINT	16#0000_0000_0000_0000 to 16#9999_9999_9999_9999 (BCD)	0 to 9999999999999999

Additional Information

- To convert a specific BCD bit string to an integer, use a **_BCD_TO_*** instruction (page 2-242).
- To convert an integer to a BCD bit string, use a **_TO_BCD_*** instruction (page 2-245).

Precautions for Correct Use

- Always use the correct instruction name for the data type of *Out*.
- If the data size of *Out* is larger than the data size of *In*, the upper digits of *Out* will contain 0.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of *In* is outside of the valid range.
  - The value in *In* is not BCD bit string data (i.e., contains A, B, C, D, E, or F hexadecimal).

BCDsToBin

The BCDsToBin instruction converts signed BCD bit strings to signed integers.

Instruction	Name	FB/FUN	Graphic expression	ST expression
BCDsToBin	Signed BCD-to-Signed Integer Conversion	FUN	(@)BCDsToBin EN ENO In Out Format	Out:=BCDsToBin(In, Format);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*1	---	*2
Format	Data format number	Input	Format of BCD bit string	_BCD0 to _BCD3	---	_BCD0
Out	Conversion result	Output	Conversion result	*1	---	---

*1 The valid range depends on the value of *Format*. Refer to *Function*, below, for details.

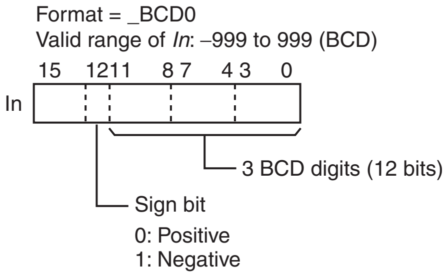
*2 If you omit the input parameter, the default value is not applied. A building error will occur.

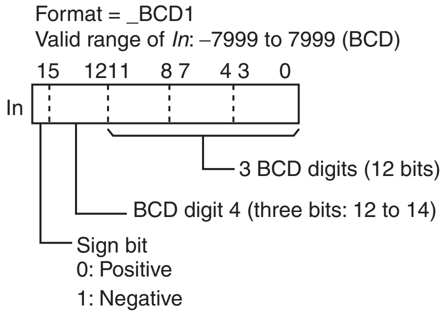
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Format	Refer to *Function* for the enumerators of the enumerated type _eBCD_FORMAT.																			
Out	Must be a signed integer data type that is the same size as *In.*																			

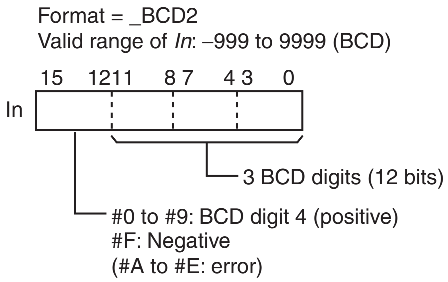
Function

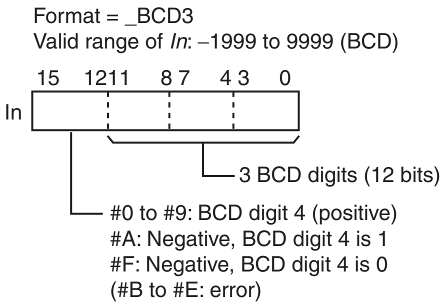
The BCDsToBin instruction converts signed BCD bit string *In* to a signed integer.

The data type of data format number *Format* is enumerated type _eBCD_FORMAT. Select one of the following: _BCD0, _BCD1, _BCD2, or _BCD3. The sign specification in the upper four bits of *In* depends on the BCD format number. The data format examples shown below use WORD data for *In.*









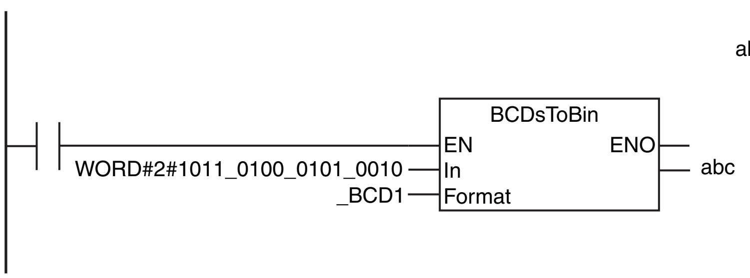
The same sizes of data types are used for *In* and *Out*. The valid ranges depend on the value of *Format*, as shown below.

		Value of *Format*			
		_BCD0	**_BCD1**	**_BCD2**	**_BCD3**
Data type of *In* ↓ Data type of *Out*	**BYTE ↓ SINT**	–9 to 9	–79 to 79	–9 to 99	–19 to 99
	WORD ↓ INT	–999 to 999	–7999 to 7999	–999 to 9999	–1999 to 9999
	DWORD ↓ DINT	–9999999 to 9999999	–79999999 to 79999999	–9999999 to 99999999	–19999999 to 99999999
	LWORD ↓ LINT	–999999999999999 to 999999999999999	–7999999999999999 to 7999999999999999	–999999999999999 to 9999999999999999	–1999999999999999 to 9999999999999999

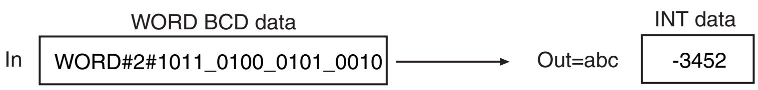
The following example is for when In is WORD#2#1011_0100_0101_0010 and *Format* is _BCD1.

LD

ST




```
abc:=BCDsToBin(WORD#2#1011_0100_0101_0010,
               _BCD1);
```

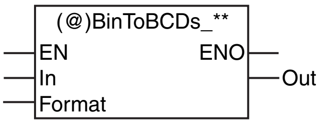


Precautions for Correct Use

- Use the same sizes of data types for *In* and *Out*.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of *Format* is _BCD0 and the upper digit of *In* is 2 to F.
  - The value of *Format* is _BCD2 and the upper digit of *In* is A to E.
  - The value of *Format* is _BCD3 and the upper digit of *In* is B to E.
  - Except for the above conditions, any digit in *In* is A to F.
  - The value of *Format* is outside of the valid range.

BinToBCDs_**

These instructions convert signed integers to signed BCD bit strings.

Instruction	Name	FB/FUN	Graphic expression	ST expression
BinToBCDs_**	Signed Integer-to-BCD Conversion Group	FUN	 "**" must be a bit string data type.	Out:=BinToBCDs(In, Format); "**" must be a bit string data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*	---	0
Format	Data format number		Format of BCD bit string	_BCD0 to _BCD3		_BCD0
Out	Conversion result	Output	Conversion result	*	---	---

* The valid range depends on the value of *Format*. Refer to *Function*, below, for details.

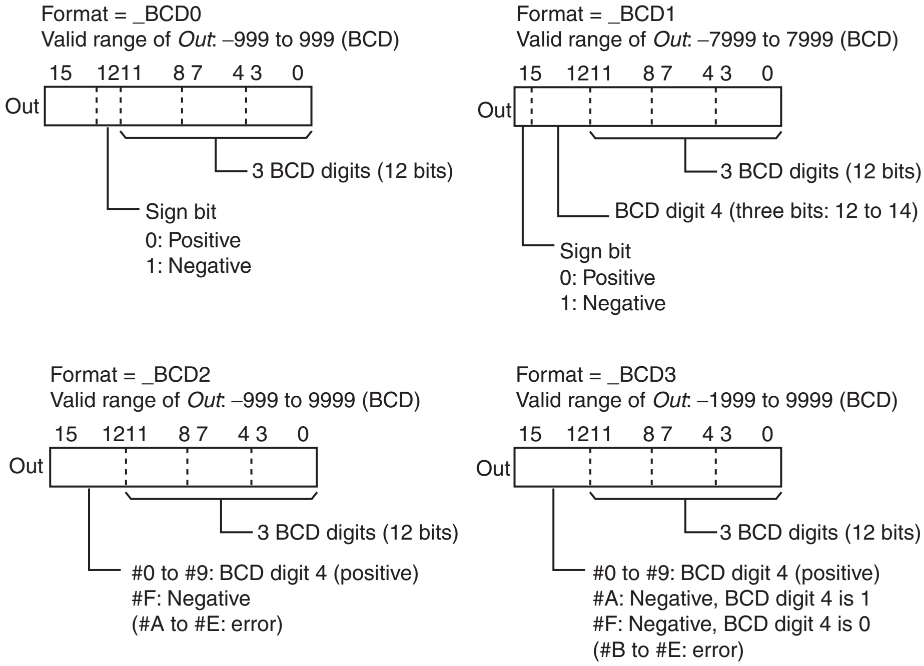
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In										OK	OK	OK	OK							
Format	Refer to *Function* for the enumerators of the enumerated type _eBCD_FORMAT.																			
Out	Must be same size of data type as *In*																			

Function

These instructions convert signed integer *In* to a signed BCD bit string.

The name of the instruction is determined by the data type of *Out*. For example, if *Out* is the WORD data type, the instruction is BinToBCDs_WORD.

The data type of data format number *Format* is enumerated type _eBCD_FORMAT. Select one of the following: _BCD0, _BCD1, _BCD2, or _BCD3. The sign specification in the upper four bits of *Out* depends on the BCD format number. The data format examples shown below use WORD data for *Out.*

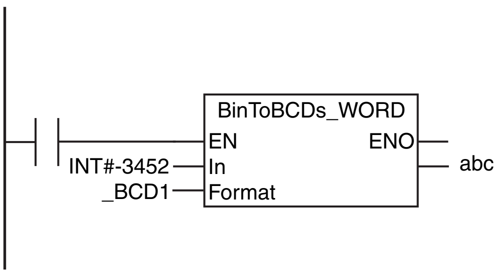


The same sizes of data types are used for *In* and *Out*. The valid ranges depend on the value of *Format*, as shown below.

		Value of *Format*			
		_BCD0	**_BCD1**	**_BCD2**	**_BCD3**
Data type of *In* ↓ Data type of *Out*	**SINT ↓ BYTE**	–9 to 9	–79 to 79	–9 to 99	–19 to 99
	INT ↓ WORD	–999 to 999	–7999 to 7999	–999 to 9999	–1999 to 9999
	DINT ↓ DWORD	–9999999 to 9999999	–79999999 to 79999999	–9999999 to 99999999	–19999999 to 99999999
	LINT ↓ LWORD	–999999999999999 to 999999999999999	–7999999999999999 to 7999999999999999	–999999999999999 to 9999999999999999	–1999999999999999 to 9999999999999999

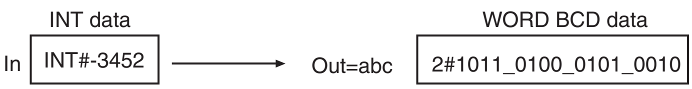
The following example shows the BinToBCDs_WORD instruction when *In* is INT#–3452 and *Format* is _BCD1.

LD



ST

```
abc:=BinToBCDs_WORD(INT#-3452,_BCD1);
```

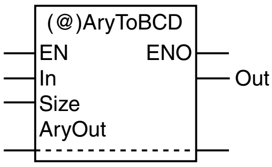


Precautions for Correct Use

- Always use the correct instruction name for the data type of *Out*.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of *In* is outside of the valid range.
  - The value of *Format* is outside of the valid range.

AryToBCD

The AryToBCD instruction converts the elements of an unsigned integer array to BCD bit strings.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryToBCD	Array BCD Conversion	FUN		AryToBCD(In, Size, AryOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Unsigned integer array	Input	Unsigned integer array	*1	---	*2
Size	Number of elements	Input	Number of elements of *In[]* for conversion	Depends on data type.	---	1
AryOut[] (array)	BCD array	In-out	BCD array	*1	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

*1 The valid ranges depend on the data types of the elements of *In[]* and *AryOut[]*. Refer to *Function* for details.

*2 If you omit an input parameter, the default value is not applied. A building error will occur.

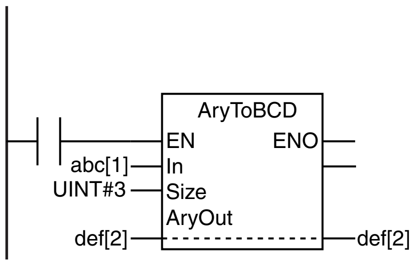
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)						OK	OK	OK	OK											
Size							OK													
AryOut[] (array)	Must be a bit string array. The data type must be the same size as the elements of *In[]*.																			
Out	OK																			

Function

The AryToBCD instruction converts *Size* elements of unsigned integer array *In[]* starting from *In[0]* to a BCD bit string. It outputs the BCD bit string to BCD array *AryOut[]*.

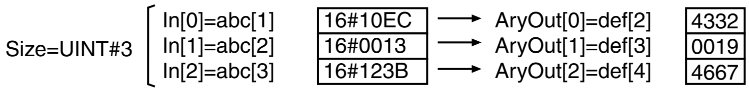
The following example is for when *Size* is UINT#3.

LD



ST

```
AryToBCD(abc[1], UINT#3, def[2]);
```



The following table shows the valid ranges for *In[]* and *AryOut[]* according to the data types of their elements.

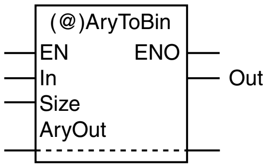
Data type of the elements of *In[]*	Data type of the elements of *AryOut[]*	Valid range of *In[]*	Valid range of *AryOut[]*
USINT	BYTE	0 to 99	16#00 to 16#99 (BCD)
UINT	WORD	0 to 9999	16#0000 to 16#9999 (BCD)
UDINT	DWORD	0 to 99999999	16#0000_0000 to 16#9999_9999 (BCD)
ULINT	LWORD	0 to 9999999999999999	16#0000_0000_0000_0000 to 16#9999_9999_9999_9999 (BCD)

Precautions for Correct Use

- Use the same data type and size for *In[]* and *AryOut[]*. For example, if the elements of *In[]* are UINT data, use WORD as the data type of the elements of *AryOut[]*. Otherwise, a building error will occur.
- This instruction does not convert signed binary to signed BCD. Use an unsigned integer (USINT, UINT, UDINT, or ULINT) as the data type of *In[].*
- The values in *AryOut[]* do not change if the value of *Size* is 0.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *AryOut[]* will not change.
  - The value of *In[]* is outside of the valid range.
  - The value of *Size* exceeds the array area of *In[]* or *AryOut[]*.

AryToBin

The AryToBin instruction converts the elements of an array of BCD bit strings into unsigned integers.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryToBin	Array Unsigned Integer Conversion	FUN		AryToBin(In, Size, AryOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array of BCD bit strings	Input	Array of BCD bit strings	*1	---	*2
Size	Number of elements		Number of elements of *In[]* for conversion	Depends on data type.		1
AryOut[] (array)	Unsigned integer array	In-out	Unsigned integer array	*1	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

*1 The valid ranges depend on the data types of the elements of *In[]* and *AryOut[]*. Refer to *Function* for details.

*2 If you omit an input parameter, the default value is not applied. A building error will occur.

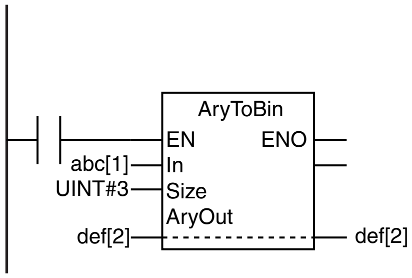
	Boolean	Bit string				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)		OK	OK	OK	OK															
Size							OK													
AryOut[] (array)	Must be an unsigned integer array. The data type must be the same size as the elements of *In[]*.																			
Out	OK																			

Function

The AryToBin instruction converts *Size* elements of array of BCD bit strings *In[]* starting from *In[0]* to unsigned integers. It outputs the unsigned integers to unsigned integer array *AryOut[]*.

The following example is for when *Size* is UINT#3.

LD



ST

```
AryToBin(abc[1], UINT#3, def[2]);
```

Size=UINT#3	In[0]=abc[1]	4332	AryOut[0]=def[2]	16#10EC
	In[1]=abc[2]	0019	AryOut[1]=def[3]	16#0013
	In[2]=abc[3]	4667	AryOut[2]=def[4]	16#123B

The following table shows the valid ranges for *In[]* and *AryOut[]* according to the data types of their elements.

Data type of the elements of *In[]*	Data type of the elements of *AryOut[]*	Valid range of *In[]*	Valid range of *AryOut[]*
BYTE	USINT	16#00 to 16#99 (BCD)	0 to 99
WORD	UINT	16#0000 to 16#9999 (BCD)	0 to 9999
DWORD	UDINT	16#0000_0000 to 16#9999_9999 (BCD)	0 to 99999999
LWORD	ULINT	16#0000_0000_0000_0000 to 16#9999_9999_9999_9999 (BCD)	0 to 9999999999999999

Precautions for Correct Use

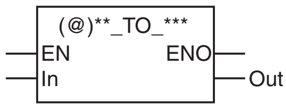
- Use the same data type and size for *In[]* and *AryOut[]*. For example, if the elements of *In[]* are WORD data, use USINT as the data type of the elements of *AryOut[]*. Otherwise, a building error will occur.
- This instruction does not convert signed BCD to signed binary. Use an unsigned integer (USINT, UINT, UDINT, or ULINT) as the data type of AryOut *[]*.
- The values in *AryOut[]* do not change if the value of *Size* is 0.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *AryOut[]* will not change.
  - The value of *Size* exceeds the array area of *In[]* or *AryOut[]*.
  - A value in In[] is not a BCD bit string (i.e., contains A, B, C, D, E, or F hexadecimal).

Data Type Conversion Instructions

Instruction	Name	Page	Instruction	Name	Page
_TO_* (Integer-to-Integer Conversion Group)	Integer-to-Integer Conversion Group	2-262	RealToFormatString	REAL-to-Formatted Text String	2-289
_TO_* (Integer-to-Bit String Conversion Group)	Integer-to-Bit String Conversion Group	2-265	LrealToFormatString	LREAL-to-Formatted Text String	2-294
_TO_* (Integer-to-Real Number Conversion Group)	Integer-to-Real Number Conversion Group	2-268	STRING_TO_** (Text String-to-Integer Conversion Group)	Text String-to-Integer Conversion Group	2-299
_TO_* (Bit String-to-Integer Conversion Group)	Bit String-to-Integer Conversion Group	2-270	STRING_TO_** (Text String-to-Bit String Conversion Group)	Text String-to-Bit String Conversion Group	2-301
_TO_* (Bit String-to-Bit String Conversion Group)	Bit String-to-Bit String Conversion Group	2-272	STRING_TO_** (Text String-to-Real Number Conversion Group)	Text String-to-Real Number Conversion Group	2-303
_TO_* (Bit String-to-Real Number Conversion Group)	Bit String-to-Real Number Conversion Group	2-274	TO_** (Integer Conversion Group)	Integer Conversion Group	2-306
_TO_* (Real Number-to-Integer Conversion Group)	Real Number-to-Integer Conversion Group	2-276	TO_** (Bit String Conversion Group)	Bit String Conversion Group	2-308
_TO_* (Real Number-to-Bit String Conversion Group)	Real Number-to-Bit String Conversion Group	2-279	TO_** (Real Number Conversion Group)	Real Number Conversion Group	2-310
_TO_* (Real Number-to-Real Number Conversion Group)	Real Number-to-Real Number Conversion Group	2-281	EnumToNum	Enumeration-to-Integer	2-312
**_TO_STRING (Integer-to-Text String Conversion Group)	Integer-to-Text String Conversion Group	2-283	NumToEnum	Integer-to-Enumeration	2-314
**_TO_STRING (Bit String-to-Text String Conversion Group)	Bit String-to-Text String Conversion Group	2-285	TRUNC, Round, and RoundUp	Truncate/Round Off Real Number/Round Up Real Number	2-316
**_TO_STRING (Real Number-to-Text String Conversion Group)	Real Number-to-Text String Conversion Group	2-287			

_TO_* (Integer-to-Integer Conversion Group)

These instructions convert integers to integers with different data types.

Instruction	Name	FB/FUN	Graphic expression	ST expression
_TO_*	Integer-to-Integer Conversion Group	FUN	 "**" and "***" must be different integer data types.	Out:=**_TO_*** (In); "**" and "***" must be different integer data types.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*	---	0
Out	Conversion result	Output	Conversion result	*	---	---

* The valid ranges depend on the data types of *In* and *Out*. Refer to *Function*, below, for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						OK	OK	OK	OK	OK	OK	OK	OK							
Out						OK	OK	OK	OK	OK	OK	OK	OK							

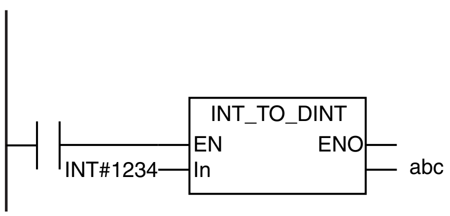
Function

These instructions convert an integer, *In*, to an integer with a different data type.

The name of the instruction is determined by the data types of *In* and conversion result *Out*. For example, if *In* is INT data and *Out* is DINT data, the name of the instruction is INT_TO_DINT.

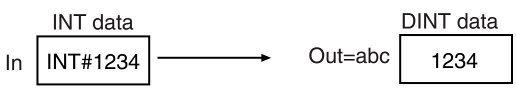
The following example for the INT_TO_DINT instruction is for when *In* is INT#1234.

LD



ST

abc:=INT_TO_DINT(INT#1234);



The following table shows the valid ranges for *In* and *Out* according to their data types.

Data type of *In*	Data type of *Out*	Valid range for *In* and *Out*
USINT	UINT	0 to 255
	UDINT	
	ULINT	
	SINT	0 to 127
	INT	0 to 255
	DINT	
	LINT	
UINT	USINT	0 to 255
	UDINT	0 to 65535
	ULINT	
	SINT	0 to 127
	INT	0 to 32767
	DINT	0 to 65535
	LINT	
UDINT	USINT	0 to 255
	UINT	0 to 65535
	ULINT	0 to 4294967295
	SINT	0 to 127
	INT	0 to 32767
	DINT	0 to 2147483647
	LINT	0 to 4294967295
ULINT	USINT	0 to 255
	UINT	0 to 65535
	UDINT	0 to 4294967295
	SINT	0 to 127
	INT	0 to 32767
	DINT	0 to 2147483647
	LINT	0 to 9223372036854775807
SINT	USINT	0 to 127
	UINT	
	UDINT	
	ULINT	
	INT	–128 to 127
	DINT	
	LINT	
INT	USINT	0 to 255
	UINT	0 to 32767
	UDINT	
	ULINT	
	SINT	–128 to 127
	DINT	–32768 to 32767
	LINT	
DINT	USINT	0 to 255
	UINT	0 to 65535
	UDINT	0 to 2147483647
	ULINT	
	SINT	–128 to 127
	INT	–32768 to 32767
	LINT	–2147483648 to 2147483647

Data type of *In*	Data type of *Out*	Valid range for *In* and *Out*
LINT	USINT	0 to 255
	UINT	0 to 65535
	UDINT	0 to 4294967295
	ULINT	0 to 9223372036854775807
	SINT	–128 to 127
	INT	–32768 to 32767
	DINT	–2147483648 to 2147483647

Additional Information

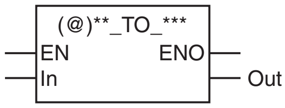
To convert data with any data type to integer data, use a TO_** (Integer Conversion Group) instruction (page 2-306).

Precautions for Correct Use

- Always use the correct instruction name for the data types of *In* and *Out*.
- If *In* is a signed integer and the data size of *Out* is larger than the data size of *In*, sign extension is performed.
- If *In* is an unsigned integer and the data size of *Out* is larger than the data size of *In*, the upper digits of *Out* will contain 0.
- If the data size of *Out* is smaller than the data size of *In*, the upper digits are truncated in *Out*.

_TO_* (Integer-to-Bit String Conversion Group)

These instructions convert integers to bit strings.

Instruction	Name	FB/FUN	Graphic expression	ST expression
_TO_*	Integer-to-Bit String Conversion Group	FUN	 "***" must be an integer data type. "****" must be a bit string data type.	Out:=**_TO_*** (In); "***" must be an integer data type. "****" must be a bit string data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*	---	0
Out	Conversion result	Output	Conversion result	*	---	---

* The valid ranges depend on the data types of *In* and *Out*. Refer to *Function*, below, for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						OK	OK	OK	OK	OK	OK	OK	OK*1							
Out		OK	OK*1	OK	OK															

*1 For an NX1P2 CPU Unit, a CPU Unit with unit version 1.14 or later and Sysmac Studio version 1.18 or higher are required to use the LINT_TO_WORD instruction.

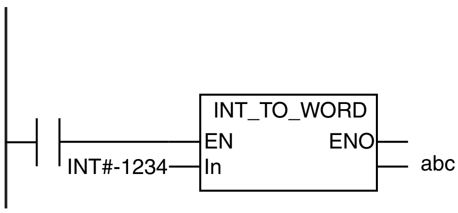
Function

These instructions convert an integer, *In*, to a bit string.

The name of the instruction is determined by the data types of *In* and conversion result *Out*. For example, if *In* is INT data and *Out* is WORD data, the name of the instruction is INT_TO_WORD.

The following example for the INT_TO_WORD instruction is for when *In* is INT#–1234.

LD



ST

```
abc:=INT_TO_WORD(INT#-1234);
```

INT data WORD data

In INT#-1234 → Out=abc 16#FB2E

The following table shows the valid ranges for *In* and *Out* according to their data types.

Data type of *In*	Data type of *Out*	Valid range for *In*	Valid range for *Out*
USINT	BYTE	0 to 255	16#00 to 16#FF
	WORD		
	DWORD		
	LWORD		
UINT	BYTE	0 to 255	16#00 to 16#FF
	WORD	0 to 65535	16#0000 to 16#FFFF
	DWORD		
	LWORD		
UDINT	BYTE	0 to 255	16#00 to 16#FF
	WORD	0 to 65535	16#0000 to 16#FFFF
	DWORD	0 to 4294967295	16#0000_0000 to 16#FFFF_FFFF
	LWORD		
ULINT	BYTE	0 to 255	16#00 to 16#FF
	WORD	0 to 65535	16#0000 to 16#FFFF
	DWORD	0 to 4294967295	16#0000_0000 to 16#FFFF_FFFF
	LWORD	0 to 18446744073709551645	16#0000_0000_0000_0000 to 16#FFFF_FFFF_FFFF_FFFF
SINT	BYTE	–128 to 127	16#00 to 16#FF
	WORD		
	DWORD		
	LWORD		
INT	BYTE	–128 to 127	16#00 to 16#FF
	WORD	–32768 to 32767	16#0000 to 16#FFFF
	DWORD		
	LWORD		
DINT	BYTE	–128 to 127	16#00 to 16#FF
	WORD	–32768 to 32767	16#0000 to 16#FFFF
	DWORD	–2147483648 to 2147483647	16#0000_0000 to 16#FFFF_FFFF
	LWORD		
LINT	BYTE	–128 to 127	16#00 to 16#FF
	WORD	–32768 to 32767	16#0000 to 16#FFFF
	DWORD	–2147483648 to 2147483647	16#0000_0000 to 16#FFFF_FFFF
	LWORD	–9223372036854775808 to 9223372036854775807	16#0000_0000_0000_0000 to 16#FFFF_FFFF_FFFF_FFFF

Additional Information

- To convert a bit string to an integer, use a **_TO_*** (Bit String-to-Integer Conversion Group) instruction (page 2-270).
- To convert data with any data type to a bit string, use a TO_** (Bit String Conversion Group) instruction (page 2-308).

Precautions for Correct Use

- Always use the correct instruction name for the data types of *In* and *Out*.
- If *In* is a signed integer and the data size of *Out* is larger than the data size of *In*, sign extension is performed.

- If *In* is an unsigned integer and the data size of *Out* is larger than the data size of *In*, the upper digits of *Out* will contain 0.
- If the data size of *Out* is smaller than the data size of *In*, the upper digits are truncated in *Out*.

_TO_* (Integer-to-Real Number Conversion Group)

These instructions convert integers to real numbers.

Instruction	Name	FB/FUN	Graphic expression	ST expression
_TO_*	Integer-to-Real Number Conversion Group	FUN	(@)**_TO_*** EN ENO In Out "**" must be an integer data type. "***" must be a real number data type.	Out:=**_TO_*** (In); "**" must be an integer data type. "***" must be a real number data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*	---	0
Out	Conversion result	Output	Conversion result	*	---	---

* The valid ranges depend on the data types of *In* and *Out*. Refer to *Function*, below, for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						OK	OK	OK	OK	OK	OK	OK	OK							
Out														OK	OK					

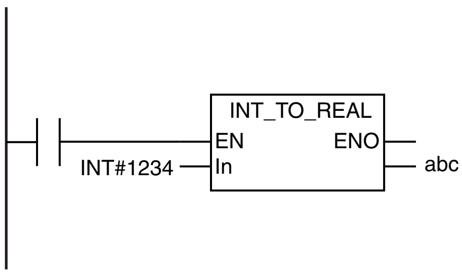
Function

These instructions convert an integer, *In*, to a real number.

The name of the instruction is determined by the data types of *In* and conversion result *Out*. For example, if *In* is INT data and *Out* is REAL data, the name of the instruction is INT_TO_REAL.

The following example for the INT_TO_REAL instruction is for when *In* is INT#1234.

LD



ST

abc:=INT_TO_REAL(INT#1234);

INT data — In: INT#1234 → Out=abc: 1.234e+3 — REAL data

The following table shows the valid ranges for *In* and *Out* according to their data types.

Data type of *In*	Data type of *Out*	Valid range for *In*	Valid range for *Out*
USINT	REAL	0 to 255	0 to 2.55e+2
	LREAL		
UINT	REAL	0 to 65535	0 to 6.5535e+4
	LREAL		
UDINT	REAL	0 to 4294967295	0 to 4.294967e+9
	LREAL		0 to 4.294967295e+9
ULINT	REAL	0 to 18446744073709551615	0 to 1.844674e+19
	LREAL		0 to 1.84467440737095e+19
SINT	REAL	–128 to 127	–1.28e+2 to 1.27e+2
	LREAL		
INT	REAL	–32768 to 32767	–3.2768e+4 to 3.2767e+4
	LREAL		
DINT	REAL	–2147483648 to 2147483647	–2.147483e+9 to 2.147483e+9
	LREAL		–2.147483648e+9 to 2.147483647e+9
LINT	REAL	–9223372036854775808 to 9223372036854775807	–9.223372e+18 to 9.223372e+18
	LREAL		–9.22337203685477e+18 to 9.22337203685477e+18

Additional Information

- To convert a real number to an integer, use a **_TO_*** (Real Number-to-Integer Conversion Group) instruction (page 2-276).
- To convert data with any data type to a real number, use a TO_** (Real Number Conversion Group) instruction (page 2-310).

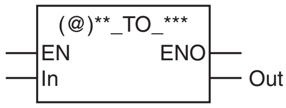
Precautions for Correct Use

- Always use the correct instruction name for the data types of *In* and *Out*.
- Depending on the data types of *In* and *Out*, rounding will be performed for the effective digits of the real number. This will cause error between the values before and after conversion. The following table lists the data types that result in error.

Data type of *In*	Data type of *Out*	Values for which error occurs
DINT	REAL	–16777216 or lower, or 16777216 or higher
LINT		
UDINT	REAL	16777216 or higher
ULINT		
LINT	LREAL	–9007199254740992 or lower, or 9007199254740992 or higher
ULINT	LREAL	9007199254740992 or higher

_TO_* (Bit String-to-Integer Conversion Group)

These instructions convert bit strings to integers.

Instruction	Name	FB/FUN	Graphic expression	ST expression
_TO_*	Bit String-to-Integer Conversion Group	FUN	 "**" must be a bit string data type. "***" must be an integer data type.	Out:=**_TO_*** (In); "**" must be a bit string data type. "***" must be an integer data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*	---	0
Out	Conversion result	Output	Conversion result	*	---	---

* The valid ranges depend on the data types of *In* and *Out*. Refer to *Function*, below, for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Out						OK	OK	OK	OK	OK	OK	OK	OK							

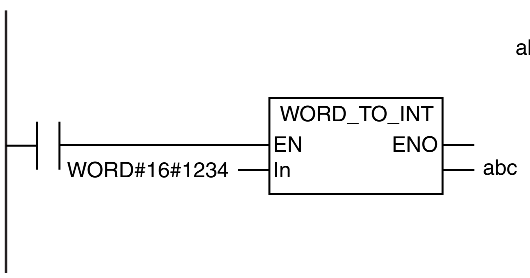
Function

These instructions convert a bit string, *In*, to an integer.

The name of the instruction is determined by the data types of *In* and conversion result *Out*. For example, if *In* is WORD data and *Out* is INT data, the name of the instruction is WORD_TO_INT.

The following example for the WORD_TO_INT instruction is for when *In* is WORD #16#1234.

LD



ST

abc:=WORD_TO_INT(WORD#16#1234);

	WORD data			INT data
In	WORD#16#1234	→	Out=abc	4660

The following table shows the valid ranges for *In* and *Out* according to their data types.

Data type of *In*	Data type of *Out*	Valid range for *In*	Valid range for *Out*
BYTE	USINT	16#00 to 16#FF	0 to 255
	UINT		
	UDINT		
	ULINT		
	SINT		–128 to 127
	INT		
	DINT		
	LINT		
WORD	USINT	16#00 to 16#FF	0 to 255
	UINT	16#0000 to 16#FFFF	0 to 65535
	UDINT		
	ULINT		
	SINT	16#00 to 16#FF	–128 to 127
	INT	16#0000 to 16#FFFF	–32768 to 32767
	DINT		
	LINT		
DWORD	USINT	16#00 to 16#FF	0 to 255
	UINT	16#0000 to 16#FFFF	0 to 65535
	UDINT	16#0000_0000 to 16#FFFF_FFFF	0 to 4294967295
	ULINT		
	SINT	16#00 to 16#FF	–128 to 127
	INT	16#0000 to 16#FFFF	–32768 to 32767
	DINT	16#0000_0000 to 16#FFFF_FFFF	–2147483648 to 2147483647
	LINT		
LWORD	USINT	16#00 to 16#FF	0 to 255
	UINT	16#0000 to 16#FFFF	0 to 65535
	UDINT	16#0000_0000 to 16#FFFF_FFFF	0 to 4294967295
	ULINT	16#0000_0000_0000_0000 to 16#FFFF_FFFF_FFFF_FFFF	0 to 18446744073709551645
	SINT	16#00 to 16#FF	–128 to 127
	INT	16#0000 to 16#FFFF	–32768 to 32767
	DINT	16#0000_0000 to 16#FFFF_FFFF	–2147483648 to 2147483647
	LINT	16#0000_0000_0000_0000 to 16#FFFF_FFFF_FFFF_FFFF	-9223372036854775808 to 9223372036854775807

Additional Information

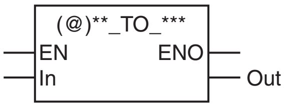
- To convert an integer to a bit string, use a **_TO_*** (Integer-to-Bit String Conversion Group) instruction (page 2-265).
- To convert data with any data type to a bit string, use a TO_** (Bit String Conversion Group) instruction (page 2-308).

Precautions for Correct Use

- Always use the correct instruction name for the data types of *In* and *Out*.
- If the data size of *Out* is larger than the data size of *In*, the upper digits of *Out* will contain 0.
- If the data size of *Out* is smaller than the data size of *In*, the upper digits are truncated in *Out*.

_TO_* (Bit String-to-Bit String Conversion Group)

These instructions convert bit strings to bit strings with different data types.

Instruction	Name	FB/FUN	Graphic expression	ST expression
_TO_*	Bit String-to-Bit String Conversion Group	FUN	 "***" and "****" must be different bit string data types.	Out:=**_TO_*** (In); "***" and "****" must be different bit string data types.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*	---	0
Out	Conversion result	Output	Conversion result	*	---	---

* The valid ranges depend on the data types of *In* and *Out*. Refer to *Function*, below, for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Out		OK	OK	OK	OK															

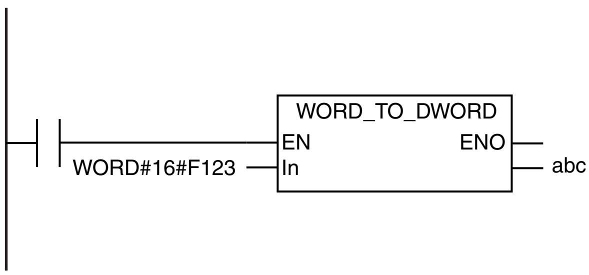
Function

These instructions convert a bit string, *In*, to a bit string with a different data type.

The name of the instruction is determined by the data types of *In* and conversion result *Out*. For example, if *In* is WORD data and *Out* is DWORD data, the name of the instruction is WORD_TO_DWORD.

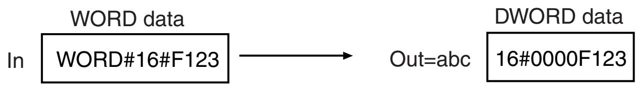
The following example for the WORD_TO_DWORD instruction is for when *In* is WORD#16#F123.

LD



ST

abc:=WORD_TO_DWORD(WORD#16#F123);



The following table shows the valid ranges for *In* and *Out* according to their data types.

Data type of *In*	Data type of *Out*	Valid range for *In* and *Out*
BYTE	WORD	16#00 to 16#FF
	DWORD	
	LWORD	
WORD	BYTE	16#00 to 16#FF
	DWORD	16#0000 to 16#FFFF
	LWORD	
DWORD	BYTE	16#00 to 16#FF
	WORD	16#0000 to 16#FFFF
	LWORD	16#0000_0000 to 16#FFFF_FFFF
LWORD	BYTE	16#00 to 16#FF
	WORD	16#0000 to 16#FFFF
	DWORD	16#0000_0000 to 16#FFFF_FFFF

Additional Information

To convert data with any data type to a bit string, use a TO_** (Bit String Conversion Group) instruction (page 2-308).

Precautions for Correct Use

- Always use the correct instruction name for the data types of *In* and *Out*.
- If the data size of *Out* is larger than the data size of *In*, the upper digits of *Out* will contain 0.
- If the data size of *Out* is smaller than the data size of *In*, the upper digits are truncated when the data is output to *Out*.

_TO_* (Bit String-to-Real Number Conversion Group)

These instructions convert bit strings to real numbers.

Instruction	Name	FB/FUN	Graphic expression	ST expression
_TO_*	Bit String-to-Real Number Conversion Group	FUN	(@)**_TO_*** EN ENO In Out "**" must be a bit string data type. "***" must be a real number data type.	Out:=**_TO_*** (In); "**" must be a bit string data type. "***" must be a real number data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*	---	0
Out	Conversion result	Output	Conversion result	*	---	---

* The valid ranges depend on the data types of *In* and *Out*. Refer to *Function*, below, for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Out														OK	OK					

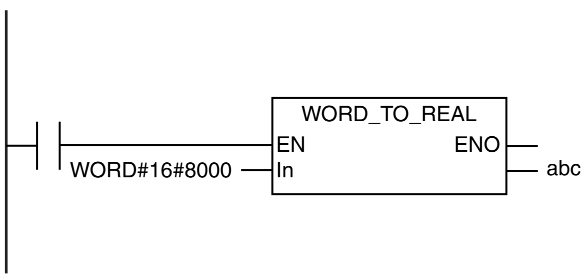
Function

These instructions take a bit string, *In*, as an unsigned integer of the same size and convert it to a real number.

The name of the instruction is determined by the data types of *In* and conversion result *Out*. For example, if *In* is WORD data and *Out* is REAL data, the name of the instruction is WORD_TO_REAL.

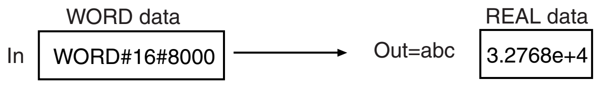
The following example for the WORD_TO_REAL instruction is for when *In* is WORD#16#8000.

LD



ST

abc:=WORD_TO_REAL(WORD#16#8000);



The following table shows the valid ranges for *In* and *Out* according to their data types.

Data type of *In*	Data type of *Out*	Valid range for *In*	Valid range for *Out*
BYTE	REAL	16#00 to 16#FF	0 to 2.55e+2
	LREAL		
WORD	REAL	16#0000 to 16#FFFF	0 to 6.5535e+4
	LREAL		
DWORD	REAL	16#0000_0000 to 16#FFFF_FFFF	0 to 4.294967e+9
	LREAL		0 to 4.294967295e+9
LWORD	REAL	16#0000_0000_0000_0000 to 16#FFFF_FFFF_FFFF_FFFF	0 to 1.844674e+19
	LREAL		0 to 1.84467440737095e+19

Additional Information

- To convert a real number to a bit string, use a **_TO_*** (Real Number-to-Bit String Conversion Group) instruction (page 2-279).
- To convert data with any data type to a real number, use a TO_** (Real Number Conversion Group) instruction (page 2-310).

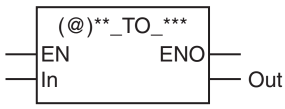
Precautions for Correct Use

- Always use the correct instruction name for the data types of *In* and *Out*.
- Depending on the data types of *In* and *Out*, rounding will be performed for the effective digits of the real number. This will cause error between the values before and after conversion. The following table lists the data types that result in error.

Data type of *In*	Data type of *Out*	Values for which error occurs
DWORD	REAL	16#0100_0000 or higher
LWORD	LREAL	16#0002_0000_0000_0000 or higher

_TO_* (Real Number-to-Integer Conversion Group)

These instructions convert real numbers to integers.

Instruction	Name	FB/FUN	Graphic expression	ST expression
_TO_*	Real Number-to-Integer Conversion Group	FUN	 "**" must be a real number data type. "***" must be an integer data type.	Out:=**_TO_*** (In); "**" must be a real number data type. "***" must be an integer data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*	---	0
Out	Conversion result	Output	Conversion result	*	---	---

* The valid ranges depend on the data types of *In* and *Out*. Refer to *Function*, below, for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK	OK					
Out						OK	OK	OK	OK	OK	OK	OK	OK							

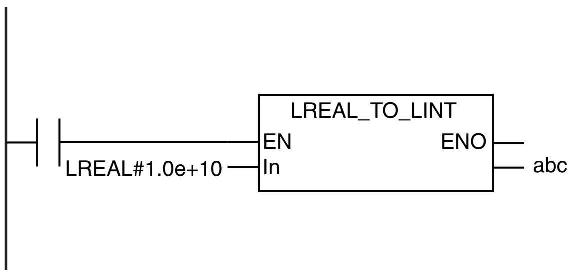
Function

These instructions convert a real number, *In*, to an integer.

The name of the instruction is determined by the data types of *In* and conversion result *Out*. For example, if *In* is LREAL data and *Out* is LINT data, the name of the instruction is LREAL_TO_LINT.

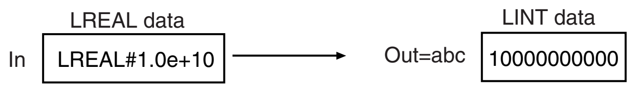
The following example for the LREAL_TO_LINT instruction is for when *In* is LREAL#1.0e+10.

LD



ST

```
abc:=LREAL_TO_LINT(LREAL#1.0e+10);
```



The fractional part of the value of *In* is rounded off to the closest integer. The following table shows how values are rounded.

Value of fractional part	Treatment	Examples
Less than 0.5	The fractional part is truncated.	1.49 → 1 –1.49 → –1
0.5	If the ones digit is an even number, the fractional part is truncated. If it is an odd number, the value is rounded up.	1.50 → 2 2.50 → 2 –1.50 → –2 –2.50 → –2
Greater than 0.5	The fractional part is rounded up.	1.51 → 2 –1.51 → –2

The following table shows the valid ranges for *In* and *Out* according to their data types.

Data type of *In*	Data type of *Out*	Valid range for *In*	Valid range for *Out*
REAL	USINT	0 to 2.55e+2	0 to 255
	UINT	0 to 6.5535e+4	0 to 65535
	UDINT	0 to 4.294967e+9	0 to 4294967295
	ULINT	0 to 1.844674e+19	0 to 18446744073709551615
	SINT	–1.28e+2 to 1.27e+2	–128 to 127
	INT	–3.2768e+4 to 3.2767e+4	–32768 to 32767
	DINT	–2.147483e+9 to 2.147483e+9	–2147483648 to 2147483647
	LINT	–9.223372e+18 to 9.223372e+18	–9223372036854775808 to 9223372036854775807
LREAL	USINT	0 to 0.255e+3	0 to 255
	UINT	0 to 6.5535e+4	0 to 65535
	UDINT	0 to 4.294967295e+9	0 to 4294967295
	ULINT	0 to 1.84467440737095e+19	0 to 18446744073709551615
	SINT	–1.28e+2 to 1.27e+2	–128 to 127
	INT	–3.2768e+4 to 3.2767e+4	–32768 to 32767
	DINT	–2.147483648e+9 to 2.147483647e+9	–2147483648 to 2147483647
	LINT	–9.22337203685477e+18 to 9.22337203685477e+18	–9223372036854775808 to 9223372036854775807

Additional Information

- To convert an integer to a real number, use an Integer-to-Real Number Conversion Group Instruction.
- To convert data with any data type to an integer, use an Integer Conversion Group Instruction.
- You can use the following instructions to convert a real number to an integer: TRUNC (Truncate), Round (Round Off Real Number), and RoundUp (Round Up Real Number). All of these instructions have a REAL input and DINT output, or a LREAL input and LINT output. The differences between these instructions are shown in the following table.

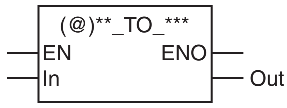
Input value	Output value			
	REAL_TO_INT	TRUNC	Round	RoundUp
REAL#1.6	INT#2	DINT#1	DINT#2	DINT#2
REAL#1.5	INT#2	DINT#1	DINT#2	DINT#2
REAL#1.5	INT#1	DINT#1	DINT#1	DINT#2
REAL#2.5	INT#2	DINT#2	DINT#2	DINT#3
REAL#-1.6	INT#-2	DINT#-1	DINT#-2	DINT#-2
REAL#-1.5	INT#-2	DINT#-1	DINT#-2	DINT#-2
REAL#-1.4	INT#-1	DINT#-1	DINT#-1	DINT#-2
REAL#-2.5	INT#-2	DINT#-2	DINT#-2	DINT#-3

Precautions for Correct Use

- Always use the correct instruction name for the data types of *In* and *Out*.
- If the conversion result exceeds the valid range of *Out*, *Out* will contain an undefined value. Always make sure that the value of *In* is within the valid range so that the conversion result will not exceed the valid range of *Out*.

_TO_* (Real Number-to-Bit String Conversion Group)

These instructions convert real numbers to bit strings.

Instruction	Name	FB/FUN	Graphic expression	ST expression
_TO_*	Real Number-to-Bit String Conversion Group	FUN	 "**" must be a real number data type. "***" must be a bit string data type.	Out:=**_TO_*** (In); "**" must be a real number data type. "***" must be a bit string data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	Depends on data type.	---	0
Out	Conversion result	Output	Conversion result	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK	OK					
Out		OK	OK	OK	OK															

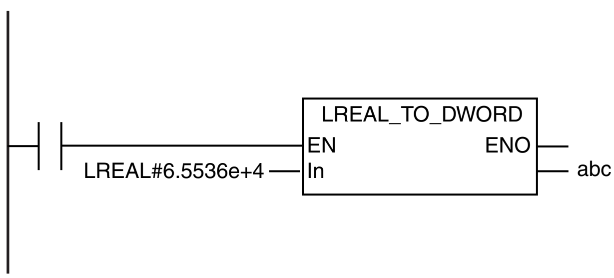
Function

These instructions convert a real number, *In*, to a bit string.

The name of the instruction is determined by the data types of *In* and conversion output *Out*. For example, if *In* is LREAL data and *Out* is DWORD data, the name of the instruction is LREAL_TO_DWORD.

The following example for the LREAL_TO_DWORD instruction is for when *In* is LREAL#6.5536e+4.

LD



ST

```
abc:=LREAL_TO_DWORD(LREAL#6.5536e+4);
```

	LREAL data			DWORD data
In	LREAL#6.5536e+4	→	Out=abc	16#0001_0000

Conversion is performed using the following procedure.

1. The fractional part of the value of *In* is rounded off to the closest integer as described below.
2. The resulting integer is taken as an unsigned integer and output as a bit string.

The following table shows how values are rounded.

Value of fractional part	Treatment	Examples
Less than 0.5	The fractional part is truncated.	1.49 → 1 –1.49 → –1
0.5	If the ones digit is an even number, the fractional part is truncated. If it is an odd number, the value is rounded up.	1.50 → 2 2.50 → 2 –1.50 → –2 –2.50 → –2
Greater than 0.5	The fractional part is rounded up.	1.51 → 2 –1.51 → –2

The following table gives some conversion examples.

Value of *In*	Integer	Value of *Out*
1.6	2	16#0002
3.5	4	16#0004
–1.6	–2	16#FFFE

The following table shows the valid ranges for *In* and *Out* according to their data types.

Data type of *In*	Data type of *Out*	Valid range for *In*	Valid range for *Out*
REAL	BYTE	–1.285999e+2 to 1.274999e+2	16#00 to 16#FF
	WORD	–3.276859e+4 to 3.276749e+4	16#0000 to 16#FFFF
	DWORD	–2.147483e+9 to 2.147483e+9	16#0000_0000 to 16#FFFF_FFFF
	LWORD	–9.223372e+18 to 9.223372e+18	16#0000_0000_0000_0000 to 16#FFFF_FFEE_FFFF_FFFF
LREAL	BYTE	–1.28599999999999e+2 to 1.27499999999999e+2	16#00 to 16#FF
	WORD	–3.27685999999999e+4 to 3.27674999999999e+4	16#0000 to 16#FFFF
	DWORD	–2.14748364859999e+9 to 2.14748364749999e+9	16#0000_0000 to 16#FFFF_FFFF
	LWORD	–9.22337203685477e+18 to 9.22337203685477e+18	16#0000_0000_0000_0000 to 16#FFFF_FFFF_FFFF_FFFF

Additional Information

To convert a bit string to a real number, use a **_TO_*** (Bit String-to-Real Number Conversion Group) instruction (page 2-274).

Precautions for Correct Use

- Always use the correct instruction name for the data types of *In* and *Out*.
- If the conversion result exceeds the valid range of *Out*, *Out* will contain an undefined value. Always make sure that the value of *In* is within the valid range so that the conversion result will not exceed the valid range of *Out*.

_TO_* (Real Number-to-Real Number Conversion Group)

These instructions convert real numbers to real numbers with different data types.

Instruction	Name	FB/FUN	Graphic expression	ST expression
_TO_*	Real Number-to-Real Number Conversion Group	FUN	(@)**_TO_*** EN ENO In Out "***" and "****" must be different real number data types.	Out:=**_TO_*** (In); "***" and "****" must be different real number data types.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*	---	0
Out	Conversion result	Output	Conversion result	*	---	---

* The valid ranges depend on the data types of In and *Out*. Refer to *Function*, below, for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK	OK					
Out														OK	OK					

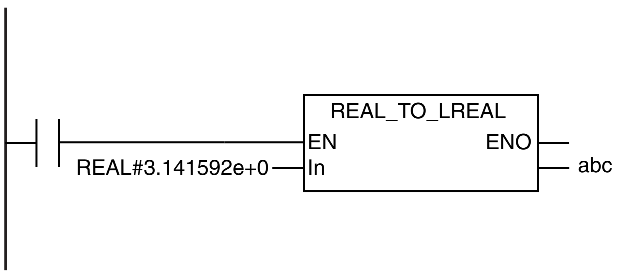
Function

These instructions convert a real number, *In*, to a real number with a different data type.

The name of the instruction is determined by the data types of *In* and conversion result *Out*. For example, if *In* is REAL data and *Out* is LREAL data, the name of the instruction is REAL_TO_LREAL.

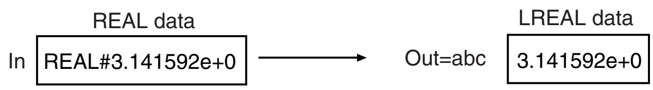
The following example for the REAL_TO_LREAL instruction is for when *In* is REAL#3.141592e+0.

LD



ST

```
abc:=REAL_TO_LREAL(REAL#3.141592e+0);
```



The following table shows the valid ranges for *In* and *Out* according to their data types.

Data type of *In*	Data type of *Out*	Valid range for *In* and *Out*
REAL	LREAL	–3.402823e+38 to 3.402823e+38 or +∞/–∞
LREAL	REAL	

Additional Information

To convert data with any data type to a real number, use a TO_** (Real Number Conversion Group) instruction (page 2-310).

Precautions for Correct Use

- Always use the correct instruction name for the data types of *In* and *Out*.
- If the value of *In* is positive or negative infinity, the value of *Out* is positive or negative infinity.
- If the value of *In* is nonnumeric data, the value of *Out* is nonnumeric data.
- If the conversion result exceeds the valid range of *Out*, the value of *Out* will be infinity with the same sign as the value of *In*.
- For the LREAL_TO_REAL instruction, if the value of *In* is closer to 0 than ±1.175494e–38, the value of *Out* will be 0.

**_TO_STRING (Integer-to-Text String Conversion Group)

These instructions convert integers to text strings.

Instruction	Name	FB/FUN	Graphic expression	ST expression
**_TO_STRING	Integer-to-Text String Conversion Group	FUN	(@)**_TO_STRING EN ENO In Out "**" must be an integer data type.	Out:=**_TO_STRING(In); "**" must be an integer data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	Depends on data type.	---	0
Out	Conversion result	Output	Conversion result	*	---	---

* The valid range depends on the data type of *In*. Refer to *Function* for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						OK	OK	OK	OK	OK	OK	OK	OK							
Out																				OK

Function

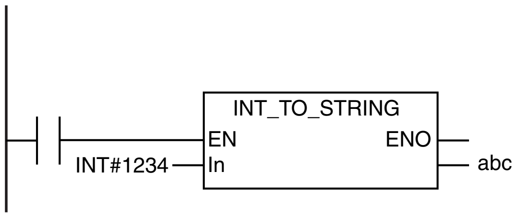
These instructions convert an integer, *In*, to a text string. The number given in *In* is output to conversion result *Out* as a text string. A NULL character (16#00) is placed at the end of *Out*.

The text in *Out* is left-aligned. If the value in *In* requires fewer digits than provided by the data type of *In*, zeros will not be output to the upper digits of *Out*. In other words, leading zeros are suppressed. If *In* contains a negative value, a minus sign (–) is added to the front of the text string.

The name of the instruction is determined by the data type of *In*. For example, if *In* is the INT data type, the instruction is INT_TO_STRING.

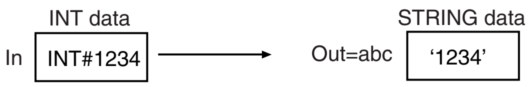
The following example for the INT_TO_STRING instruction is for when *In* is INT#1234.

LD



ST

```
abc:=INT_TO_STRING(INT#1234);
```



The valid range of *Out* depends on the data type of *In* as shown below:

Data type of *In*	Valid range of *Out* (maximum number of bytes)
USINT	4 bytes (three single-byte alphanumeric characters plus the final NULL character)
UINT	6 bytes (five single-byte alphanumeric characters plus the final NULL character)
UDINT	11 bytes (10 single-byte alphanumeric characters plus the final NULL character)
ULINT	21 bytes (20 single-byte alphanumeric characters plus the final NULL character)
SINT	5 bytes (four single-byte alphanumeric characters plus the final NULL character)
INT	7 bytes (six single-byte alphanumeric characters plus the final NULL character)
DINT	12 bytes (11 single-byte alphanumeric characters plus the final NULL character)
LINT	21 bytes (20 single-byte alphanumeric characters plus the final NULL character)

Additional Information

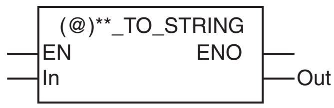
To convert a text string number to an integer, use a STRING_TO_** (Text String-to-Integer Conversion Group) instruction (page 2-299).

Precautions for Correct Use

- Always use the correct instruction name for the data type of *In*.

**_TO_STRING (Bit String-to-Text String Conversion Group)

These instructions convert bit strings to text strings.

Instruction	Name	FB/FUN	Graphic expression	ST expression
**_TO_STRING	Bit String-to-Text String Conversion Group	FUN	 "**" must be a bit string data type.	Out:=**_TO_STRING(In); "**" must be a bit string data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	Depends on data type.	---	0
Out	Conversion result	Output	Conversion result	*	---	---

* The valid range depends on the data type of *In*. Refer to *Function* for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Out																				OK

Function

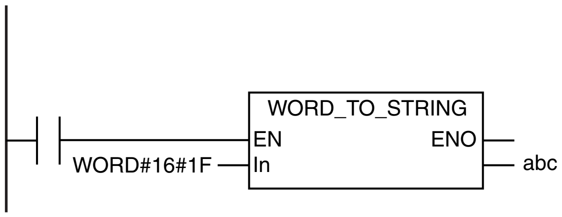
These instructions convert a bit string, *In*, to a text string. The hexadecimal number given in *In* is output to conversion result *Out* as a text string. The #16 prefix of the hexadecimal number is not output to *Out*. A NULL character (16#00) is placed at the end of *Out*.

The text in *Out* is left-aligned. If the value in *In* requires fewer digits than provided by the data type of *In*, the upper digits of *Out* will contain 0. In other words, the unused digits are padded with zeros. The number of bytes in *Out* (including the NULL character) will always be one greater than twice the number of bytes in *In*.

The name of the instruction is determined by the data type of *In*. For example, if *In* is the WORD data type, the instruction is WORD_TO_STRING.

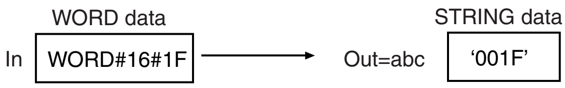
The following example for the WORD_TO_STRING instruction is for when *In* is WORD#16#1F.

LD



ST

```
abc:=WORD_TO_STRING(WORD#16#1F);
```



The valid range of *Out* depends on the data type of *In* as shown below:

Data type of *In*	Valid range of *Out* (maximum number of bytes)
BYTE	3 bytes (two single-byte alphanumeric characters plus the final NULL character)
WORD	5 bytes (four single-byte alphanumeric characters plus the final NULL character)
DWORD	9 bytes (eight single-byte alphanumeric characters plus the final NULL character)
LWORD	17 bytes (16 single-byte alphanumeric characters plus the final NULL character)

Additional Information

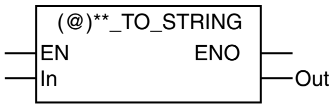
To convert *In* to a signed text string, first convert it to a signed integer using a **_TO_*** (Bit String-to-Integer Conversion Group) instruction (page 2-270) and then use a **_TO_STRING (Integer-to-Text String Conversion Group) instruction (page 2-283).

Precautions for Correct Use

- Always use the correct instruction name for the data type of *In*.

**_TO_STRING (Real Number-to-Text String Conversion Group)

These instructions convert real numbers to text strings.

Instruction	Name	FB/FUN	Graphic expression	ST expression
**_TO_STRING	Real Number-to-Text String Conversion Group	FUN	 "**" must be a real number data type.	Out:=**_TO_STRING(In); "**" must be a real number data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	Depends on data type.	---	0.0
Out	Conversion result	Output	Conversion result	*	---	---

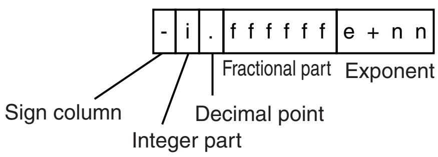
* The valid range depends on the data type of *In*. Refer to *Function* for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK	OK					
Out																				OK

Function

These instructions convert a real number, *In*, to a text string. *In* is expressed as an alphanumeric text string and output to conversion result *Out*.

The format of *Out* is as follows:



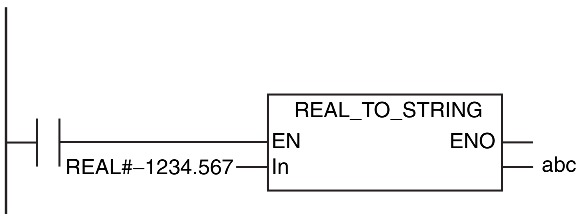
Item	Description
Sign column	If *In* contains a negative value, a minus sign (–) is added. If *In* contains a positive value, a plus sign (+) is not added.
Integer part	The integer part is always only one digit.
Decimal point	The decimal point is always given even if *In* is not a decimal number.
Fractional part	If *In* is REAL data, 6 digits are given. If *In* is LREAL data, 14 digits are given.
Exponent	The exponent is always given. “nn” is 2 or 3 digits. The sign of “nn” is positive (+) if the absolute value of *In* is 1.0 or higher and negative (–) if it is less than 1.0.

A NULL character (16#00) is placed at the end of *Out*.

The name of the instruction is determined by the data type of *In*. For example, if *In* is the REAL data type, the instruction is REAL_TO_STRING.

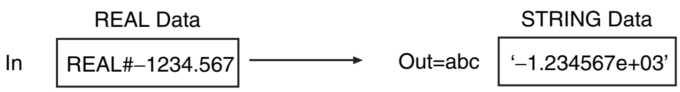
The following example shows the REAL_TO_STRING instruction when *In* is REAL#–1234.567.

LD



ST

abc:=REAL_TO_STRING(REAL#–1234.567);



If the value of *In* is 0, infinity, or nonnumeric data, the value of *Out* is as shown below.

Value of *In*	Value of *Out*
0	0
+∞	inf
–∞	–inf
Nonnumeric data	‘nan’ or ‘–nan’

Additional Information

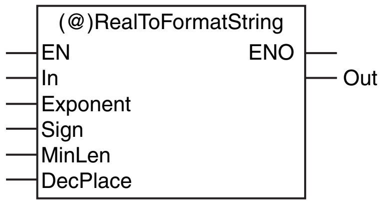
- To convert a text string to a real number, use a STRING_TO_** (Text String-to-Real Number Conversion Group) instruction (page 2-303).
- To specify the format when you convert a real number to a text string, use the RealToFormatString instruction (page 2-289) or the LrealToFormatString instruction (page 2-294).

Precautions for Correct Use

- Always use the correct instruction name for the data type of *In*.

RealToFormatString

The RealToFormatString instruction converts a REAL variable to a text string with the specified format.

Instruction	Name	FB/FUN	Graphic expression	ST expression
RealToFormatString	REAL-to-Formatted Text String	FUN		Out:=RealToFormat-String(In, Exponent, Sign, MinLen, DecPlace);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	Depends on data type.	---	0.0
Exponent	Exponent		TRUE: Exponent FALSE: No exponent			FALSE
Sign	Sign column		TRUE: Sign column FALSE: No sign column			
MinLen	Minimum number of digits		Minimum number of digits in *Out*			6
DecPlace	Precision		Number of decimal digits in *Out*	0 to 15		
Out	Conversion result	Output	Conversion result	327 bytes max. (326 single-byte alphanumeric characters plus the final NULL character)	---	---

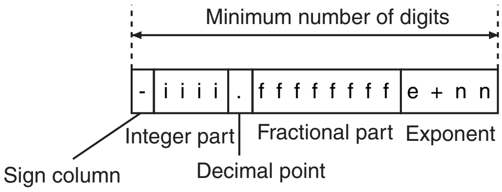
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK						
Exponent	OK																			
Sign	OK																			
MinLen						OK														
DecPlace						OK														
Out																				OK

Function

The RealToFormatString instruction converts REAL variable *In* to a text string. *In* is expressed as an alphanumeric text string and output to conversion result *Out.* A NULL character (16#00) is placed at the end of *Out*.

If *In* contains a negative value, a minus sign (–) is added to the front of the text string. If *In* contains a positive value, a plus sign (+) is not added to the front of the text string.

The format of *Out* is determined by exponent *Exponent*, sign column *Sign*, minimum number of digits *MinLen*, and precision *DecPlace*.



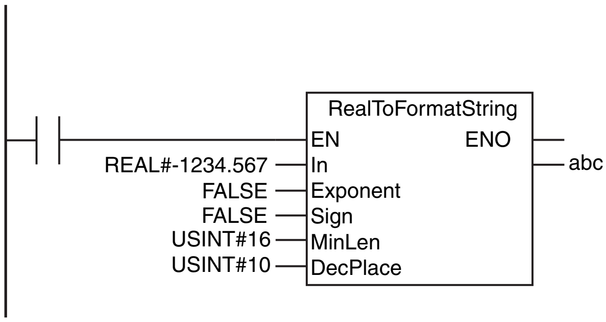
Input variable	Description
Exponent	*Exp* specifies whether an exponent is given. TRUE: Exponent FALSE: No exponent
Sign	*Sign* specifies whether there is a sign column. TRUE: Sign column FALSE: No sign column The sign column is used only for a minus sign (–). If the number is positive when the sign column is specified, the sign column will contain a blank character. If the number is negative when no sign column is specified, a minus sign (–) will be added to the front of the integer part. However, if the number of digits in the conversion result exceeds the value of *MinLen* and the conversion result is positive, the highest digit is placed in the sign column.
MinLen	*MinLen* is the minimum number of total digits for the sign column, integer part, decimal point, fractional part, and exponent. If the conversion result has fewer digits than the value of *MinLen*, the text string will be right-aligned (except for the sign column) and remaining digits will contain blank characters. If the number of digits in the conversion result exceeds the value of *MinLen*, the text string is left-aligned and the text string for the digits that exceed the value of *MinLen* is assigned to *Out*.
DecPlace	*DecPlace* is the number of digits in the fractional part. If the number of digits exceeds the value of *DecPlace*, the extra digits in the fractional portion are rounded off as described below. If the value of *DecPlace* is 0, the fractional part and decimal point are not given.

The following examples show the relationships between the values of the input variables and the value of *Out* when *In* is REAL#–1234.567.

Example 1: Exponent: FALSE
Sign: FALSE
MinLen: USINT#16
DecPlace: USINT#10

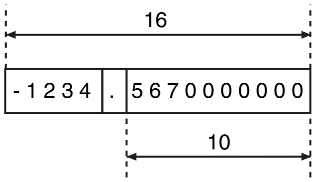
Here, no sign column is specified for a negative number, so a minus sign (–) is added to the front of the integer part.

LD



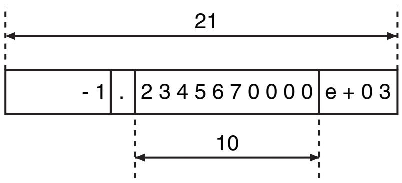
ST

```
abc:=RealToFormatString(REAL#-1234.567, FALSE,
                        FALSE, USINT#16,
                        USINT#10);
```



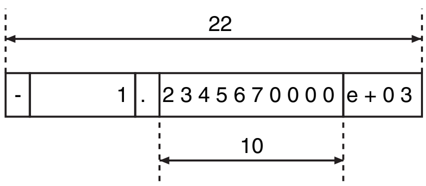
Example 2: Exponent: TRUE
Sign: FALSE
MinLen: USINT#21
DecPlace: USINT#10

Here, the value of *MinLen* exceeds the number of digits in the text string, so the text string is right-aligned and blank characters are added before it.



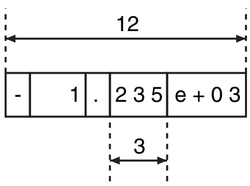
Example 3: Exponent: TRUE
Sign: TRUE
MinLen: USINT#22
DecPlace: USINT#10

The sign column is always on the left. Blank characters are added to the front of the integer part.



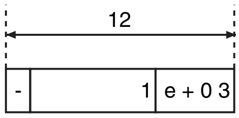
Example 4: Exponent: TRUE
Sign: TRUE
MinLen: USINT#12
DecPlace: USINT#3

The fourth decimal place is rounded off because *DecPlace* is USINT#3.



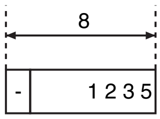
Example 5: Exponent: TRUE
Sign: TRUE
MinLen: USINT#12
DecPlace: USINT#0

The first decimal place is rounded off because *DecPlace* is USINT#0. The decimal point is also not given.



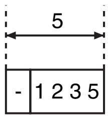
Example 6: Exponent: FALSE
Sign: TRUE
MinLen: USINT#8
DecPlace: USINT#0

Here, no exponent is given and the integer part is only four digits. The first decimal place is rounded off.



Example 7: Exponent: FALSE
Sign: TRUE
MinLen: USINT#2
DecPlace: USINT#0

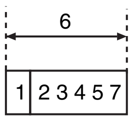
Here, the number of digits in the integer part of *In* (four digits) is larger than the value of *MinLen* (USINT#2). The four digits of the integer part are given.



The following examples show the relationships between the values of the input variables and the value of *Out* when *In* is REAL#123456.7.

Example 8: Exponent: FALSE
Sign: TRUE
MinLen: USINT#4
DecPlace: USINT#0

Here, the number of digits in the integer part of *In* (six digits) is larger than the value of *MinLen* (USINT#4). The six digits of the integer part are given. The value of *In* is positive, so the highest digit is placed in the sign column.



If the value of *In* is infinity, or nonnumeric data, the value of *Out* is as shown below.

Value of *In*	Value of *Out*
+∞	‘inf’
−∞	‘−inf’
Nonnumeric data	‘nan’ or ‘−nan’

The following table shows how values are rounded.

Value of fractional part	Treatment	Examples
Less than 0.5	The fractional part is truncated.	1.49 → 1
0.5	If the ones digit is an even number, the fractional part is truncated. If it is an odd number, the value is rounded up.	1.50 → 2 2.50 → 2
Greater than 0.5	The fractional part is rounded up.	1.51 → 2

Additional Information

- *Exponent, Sign, MinLen,* and *DecPlace* can be omitted. The defaults are applied for any omitted input variables.
- To convert a LREAL variable to a text string, use the LrealToFormatString instruction (page 2-294).
- To convert a text string to a real number, use a STRING_TO_** (Text String-to-Real Number Conversion Group) instruction (page 2-303).

Precautions for Correct Use

An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.

- The value of *DecPlace* is outside of the valid range.
- The value of *DecPlace* is greater than the value of *MinLen.*

LrealToFormatString

The LrealToFormatString instruction converts a LREAL variable to a text string with the specified format.

Instruction	Name	FB/FUN	Graphic expression	ST expression
LrealToFormatString	LREAL-to-Formatted Text String	FUN	(@)LrealToFormatString EN ENO In Out Exponent Sign MinLen DecPlace	Out:=LrealToFormatString (In, Exponent, Sign, MinLen, DecPlace);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	Depends on data type.	---	0.0
Exponent	Exponent		TRUE: Exponent FALSE: No exponent			FALSE
Sign	Sign column		TRUE: Sign column FALSE: No sign column			
MinLen	Minimum number of digits		Minimum number of digits in *Out*			6
DecPlace	Precision		Number of decimal digits in *Out*	0 to 15		
Out	Conversion result	Output	Conversion result	327 bytes max. (326 single-byte alphanumeric characters plus the final NULL character)	---	---

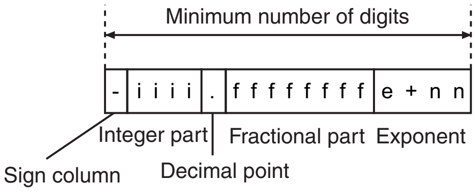
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In															OK					
Exponent	OK																			
Sign	OK																			
MinLen						OK														
DecPlace						OK														
Out																				OK

Function

The LrealToFormatString instruction converts LREAL variable *In* to a text string. *In* is expressed as an alphanumeric text string and output to conversion result *Out.* A NULL character (16#00) is placed at the end of *Out*.

If *In* contains a negative value, a minus sign (–) is added to the front of the text string. If *In* contains a positive value, a plus sign (+) is not added to the front of the text string.

The format of *Out* is determined by exponent *Exponent*, sign column *Sign*, minimum number of digits *MinLen*, and precision *DecPlace*.



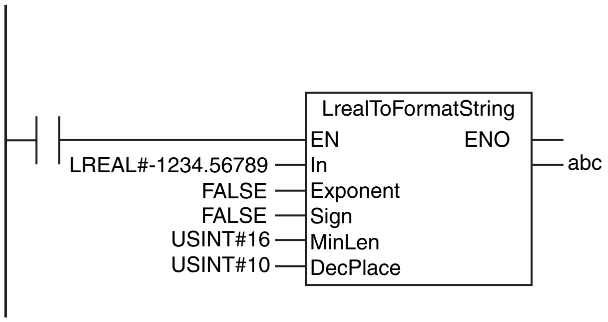
Input variable	Description
Exponent	*Exp* specifies whether an exponent is given. TRUE: Exponent FALSE: No exponent
Sign	*Sign* specifies whether there is a sign column. TRUE: Sign column FALSE: No sign column The sign column is used only for a minus sign (–). If the number is positive when the sign column is specified, the sign column will contain a blank character. If the number is negative when no sign column is specified, a minus sign (–) will be added to the front of the integer part. However, if the number of digits in the conversion result exceeds the value of *MinLen* and the conversion result is positive, the highest digit is placed in the sign column.
MinLen	*MinLen* is the minimum number of total digits for the sign column, integer part, decimal point, fractional part, and exponent. If the conversion result has fewer digits than the value of *MinLen*, the text string will be right-aligned (except for the sign column) and remaining digits will contain blank characters. If the number of digits in the conversion result exceeds the value of *MinLen*, the text string is left-aligned and the text string for the digits that exceed the value of *MinLen* is assigned to *Out*.
DecPlace	*DecPlace* is the number of digits in the fractional part. If the number of digits exceeds the value of *DecPlace*, the extra digits in the fractional portion are rounded off as described below. If the value of *DecPlace* is 0, the fractional part and decimal point are not given.

The following examples show the relationships between the values of the input variables and the value of *Out* when *In* is LREAL#–1234.56789.

Example 1: Exponent: FALSE
Sign: FALSE
MinLen: USINT#16
DecPlace: USINT#10

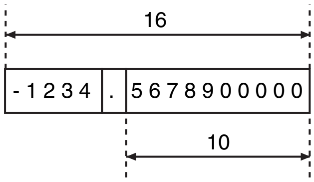
Here, no sign column is specified for a negative number, so a minus sign (–) is added to the front of the integer part.

LD



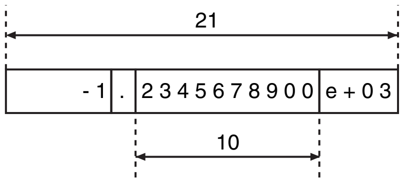
ST

```
abc:=LrealToFormatString(LREAL#-1234.56789, FALSE,
                         FALSE, USINT#16,
                         USINT#10);
```



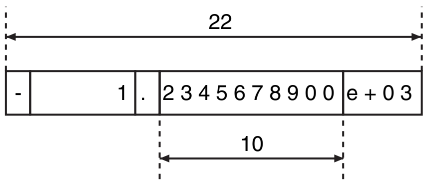
Example 2: Exponent: TRUE
Sign: FALSE
MinLen: USINT#21
DecPlace: USINT#10

Here, the value of *MinLen* exceeds the number of digits in the text string, so the text string is right-aligned and blank characters are added before it.



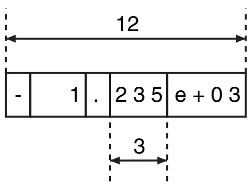
Example 3: Exponent: TRUE
Sign: TRUE
MinLen: USINT#22
DecPlace: USINT#10

The sign column is always on the left. Blank characters are added to the front of the integer part.



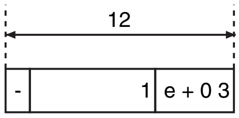
Example 4: Exponent: TRUE
Sign: TRUE
MinLen: USINT#12
DecPlace: USINT#3

The fourth decimal place is rounded off because *DecPlace* is USINT#3.



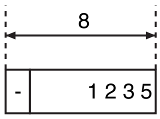
Example 5: Exponent: TRUE
Sign: TRUE
MinLen: USINT#12
DecPlace: USINT#0

The first decimal place is rounded off because *DecPlace* is USINT#0. The decimal point is also not given.



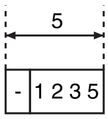
Example 6: Exponent: FALSE
Sign: TRUE
MinLen: USINT#8
DecPlace: USINT#0

Here, no exponent is given and the integer part is only four digits. The first decimal place is rounded off.



Example 7: Exponent: FALSE
Sign: TRUE
MinLen: USINT#2
DecPlace: USINT#0

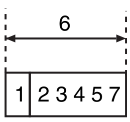
Here, the number of digits in the integer part of *In* (four digits) is larger than the value of *MinLen* (USINT#2). The four digits of the integer part are given.



The following examples show the relationships between the values of the input variables and the value of *Out* when *In* is LREAL#123456.789.

Example 8: Exponent: FALSE
Sign: TRUE
MinLen: USINT#4
DecPlace: USINT#0

Here, the number of digits in the integer part of *In* (six digits) is larger than the value of *MinLen* (USINT#4). The six digits of the integer part are given. The value of *In* is positive, so the highest digit is placed in the sign column.



If the value of In is infinity, or nonnumeric data, the value of Out is as shown below.

Value of *In*	Value of *Out*
+∞	‘inf’
−∞	‘−inf’
Nonnumeric data	‘nan’ or ‘−nan’

The following table shows how values are rounded.

Value of fractional part	Treatment	Examples
Less than 0.5	The fractional part is truncated.	1.49 → 1
0.5	If the ones digit is an even number, the fractional part is truncated. If it is an odd number, the value is rounded up.	1.50 → 2 2.50 → 2
Greater than 0.5	The fractional part is rounded up.	1.51 → 2

Additional Information

- *Exponent, Sign, MinLen,* and *DecPlace* can be omitted. The defaults are applied for any omitted input variables.
- To convert a REAL variable to a text string, use the RealToFormatString instruction (page 2-289).
- To convert a text string to a real number, use a STRING_TO_** (Text String-to-Real Number Conversion Group) instruction (page 2-303).

Precautions for Correct Use

An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.

- The value of *DecPlace* is outside of the valid range.
- The value of *DecPlace* is greater than the value of *MinLen.*

STRING_TO_** (Text String-to-Integer Conversion Group)

These instructions convert text strings to integers.

Instruction	Name	FB/FUN	Graphic expression	ST expression
STRING_TO_**	Text String-to-Integer Conversion Group	FUN	 "**" must be an integer data type.	Out:=STRING_TO_** (In); "**" must be an integer data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*	---	''
Out	Conversion result	Output	Conversion result	Depends on data type.	---	---

* The valid range depends on the data type of *Out*. Refer to *Function* for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
Out						OK	OK	OK	OK	OK	OK	OK	OK							

Function

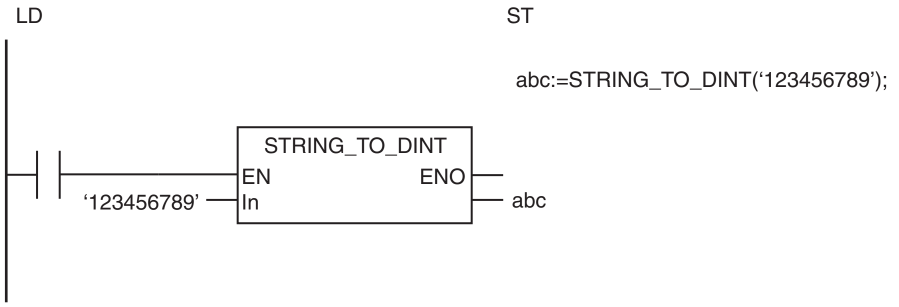
These instructions convert a text string, *In*, to an integer.

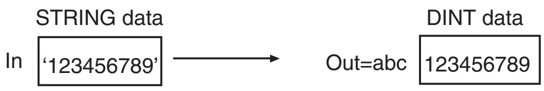
Basically, the text string in *In* must consist only of numbers 0 to 9. The following exceptions are possible.

- If the first character in *In* is a single minus sign (–) or a single plus sign (+), it is processed as the sign.
- Any blank characters at the beginning of *In* are ignored.
- Any blank characters between an initial minus sign (–) or plus sign (+) and a number are ignored.
- Any single underbars (‘_’) at any location are ignored.
- An error occurs if there are two or more consecutive underbars (‘_’) at any location.
- An error occurs if there are any underbars (‘_’) at the beginning or end.
- An error occurs if there are any underbars (‘_’) between the minus signs (‘–’) or plus sign (‘+’) and the number at the beginning.

The name of the instruction is determined by the data type of conversion result *Out*. For example, if *Out* is the DINT data type, the instruction is STRING_TO_DINT.

The following example for the STRING_TO_DINT instruction is for when *In* is '123456789'.





The valid range of *In* depends on the data type of *Out* as shown below:

Data type of *Out*	Valid range of *In* (maximum number of bytes)*
USINT	4 bytes (three single-byte alphanumeric characters plus the final NULL character)
UINT	6 bytes (five single-byte alphanumeric characters plus the final NULL character)
UDINT	11 bytes (10 single-byte alphanumeric characters plus the final NULL character)
ULINT	21 bytes (20 single-byte alphanumeric characters plus the final NULL character)
SINT	5 bytes (four single-byte alphanumeric characters plus the final NULL character)
INT	7 bytes (six single-byte alphanumeric characters plus the final NULL character)
DINT	12 bytes (11 single-byte alphanumeric characters plus the final NULL character)
LINT	21 bytes (20 single-byte alphanumeric characters plus the final NULL character)

* Any blank characters (' ') at the beginning of the text string, any zeros at the beginning of the text string, and any underbars ('_') in the text string are not included in the number of bytes.

Additional Information

- To convert a text string to a hexadecimal number, use a STRING_TO_** (Text String-to-Bit String Conversion Group) instruction (page 2-301).
- To convert an integer to a text string, use a **_TO_STRING (Integer-to-Text String Conversion Group) instruction (page 2-283).

Precautions for Correct Use

- Always use the correct instruction name for the data type of *Out*.
- If the value of *In* is '–0', the value of *Out* is 0.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The text string in *In* does not express a number.
  - The conversion result exceeds the valid range of the data type of *Out.*

STRING_TO_** (Text String-to-Bit String Conversion Group)

These instructions convert text strings to bit strings.

Instruction	Name	FB/FUN	Graphic expression	ST expression
STRING_TO_**	Text String-to-Bit String Conversion Group	FUN	(@)STRING_TO_** EN ENO In Out "**" must be a bit string data type.	Out:=STRING_TO_** (In); "**" must be a bit string data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*	---	''
Out	Conversion result	Output	Conversion result	Depends on data type.	---	---

* The valid range depends on the data type of *Out*. Refer to *Function* for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
Out		OK	OK	OK	OK															

Function

These instructions interpret the content of a text string, *In*, as a hexadecimal number and convert it to a bit string.

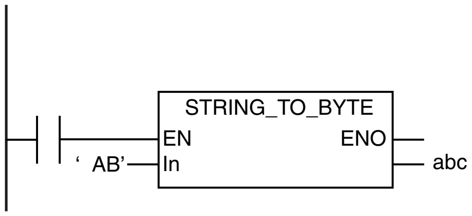
Basically, the text string in *In* must consist only of ‘0’ to ‘9’, ‘a’ to ‘f’, and ‘A’ to ‘F’. The following exception is possible.

- Any continuous blank characters or zeros at the beginning of *In* are ignored.
- Any single underbars (‘_’) at any location are ignored.
- An error occurs if there are two or more consecutive underbars (‘_’) at any location.
- An error occurs if there are any underbars (‘_’) at the beginning or end.
- An error occurs if there are any underbars (‘_’) between the minus signs (‘–’) or plus sign (‘+’) and the number at the beginning.

The name of the instruction is determined by the data type of conversion result *Out*. For example, if *Out* is the BYTE data type, the instruction is STRING_TO_BYTE.

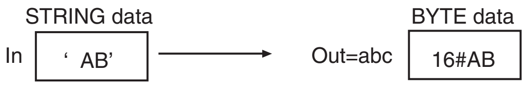
The following example for the STRING_TO_BYTE instruction is for when *In* is ‘ AB’. Any blank characters at the beginning are ignored.

LD



ST

```
abc:=STRING_TO_BYTE('  AB');
```



The valid range of *In* depends on the data type of *Out* as shown below:

Data type of *Out*	Valid range of *In* (maximum number of bytes)*
BYTE	3 bytes (two single-byte alphanumeric characters plus the final NULL character)
WORD	5 bytes (four single-byte alphanumeric characters plus the final NULL character)
DWORD	9 bytes (eight single-byte alphanumeric characters plus the final NULL character)
LWORD	17 bytes (16 single-byte alphanumeric characters plus the final NULL character)

* Any blank characters (‘ ’) at the beginning of the text string, any zeros at the beginning of the text string, and any underbars (‘_’) in the text string are not included in the number of bytes.

Additional Information

- To treat a signed number as a text string, use a STRING_TO_** (Text String-to-Integer Conversion Group) instruction (page 2-299).
- To convert a bit string to a text string, use a **_TO_STRING (Bit String-to-Text String Conversion Group) instruction (page 2-285).

Precautions for Correct Use

- Always use the correct instruction name for the data type of *Out*.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The text string in *In* does not express a number.
  - The conversion result exceeds the valid range of the data type of *Out.*

STRING_TO_** (Text String-to-Real Number Conversion Group)

These instructions convert text strings to real numbers.

Instruction	Name	FB/FUN	Graphic expression	ST expression
STRING_TO_**	Text String-to-Real Number Conversion Group	FUN	(@)STRING_TO_** — EN, In → ENO, Out "**" must be a real number data type.	Out:=STRING_TO_** (In); "**" must be a real number data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	311 bytes max. (310 single-byte alphanumeric characters plus the final NULL character)	---	''
Out	Conversion result	Output	Conversion result	Depends on data type.	---	---

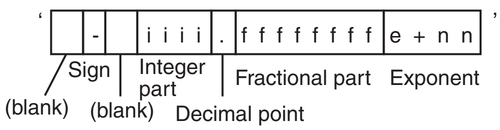
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
Out														OK	OK					

Function

These instructions convert a text string, *In*, to a real number.

The name of the instruction is determined by the data type of conversion result *Out*. For example, if *Out* is the LREAL data type, the instruction is STRING_TO_LREAL.

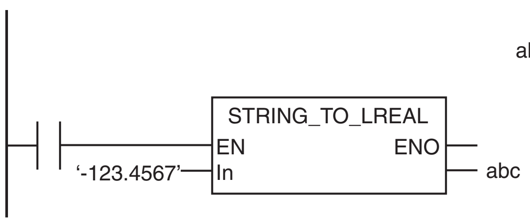
The format of the text sting in *In* is given below.



Name	Format
Sign	• Any consecutive blank characters at the beginning of the text string are ignored. Any following single plus or minus sign is treated as the sign. • The plus sign can be omitted. • Any consecutive blank characters after the sign are ignored.
Integer part	• The characters after the sign and up to the decimal point are taken as the integer part. Any consecutive blank characters after the sign are not included in the integer part. The sign may sometimes be omitted. • If the decimal point and fractional part are omitted, the characters up to the exponent are taken as the integer part. • If the decimal point, fractional part, and exponent are omitted, the characters up to the end of the text string are taken as the integer part. • The integer part consists of '0' to '9'. • The integer part cannot be omitted. • The maximum number of digits in the integer part is the maximum text string length of 1985 minus the total number of bytes in the following: the sign, decimal point, fractional part, exponent, and blank characters before and after the sign.
Decimal point	• A single period ('.') following the integer part is taken as the decimal point. • Omit the decimal point if there is no fractional part.
Fractional part	• The characters after the decimal point and up to the exponent are taken as the fractional part. • If the exponent is omitted, the characters up to the end of the text string are taken as the fractional part. • The fractional part consists of '0' to '9'. • The fractional part can be omitted. • The fractional part can consist of a maximum of 15 digits. • If there is no decimal point, then there is no fractional part.
Exponent	• The exponent consists of a single 'e' or 'E' after the fractional part, a following single plus or minus sign, and the remaining characters to the end of the text string. • If there is no fractional part, then the above text string after the decimal point is taken as the exponent. • If there is no decimal point or fractional part, then the above text string after the integer part is taken as the exponent. • The numeric part of the exponent consists of '0' to '9'. • The exponent can be omitted. • The numeric part of the exponent can consist of a maximum of three digits.

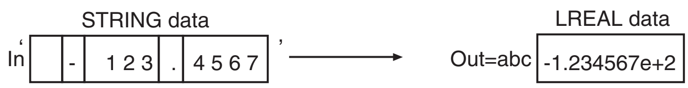
Example 1: The following example uses the sign, decimal point, and fractional part, but does not use an exponent.

LD



ST

```
abc:=STRING_TO_LREAL('-123.4567');
```



Example 2: The following example uses the sign, decimal point, fractional part, and exponent.

Example 3: The following example does not use the sign, but uses the decimal point, fractional part, and exponent.

Example 4: The following example does not use the sign, fractional part, decimal point, and exponent.

If the value of *In* is ‘+inf’, the value of *Out* is positive infinity. If the value of *In* is ‘–inf’, the value of *Out* is negative infinity. In either case, characters are not case sensitive.

Additional Information

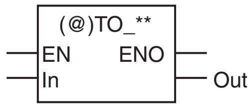
To convert a real number to a text string, use a **_TO_STRING (Real Number-to-Text String Conversion Group) instruction (page 2-287).

Precautions for Correct Use

- Always use the correct instruction name for the data type of *Out*.
- Any single underbars (‘_’) at any location in In are ignored.
- An error occurs if there are any underbars (‘_’) at the beginning or end of *In*.
- An error occurs if there are two or more consecutive underbars (‘_’) at any location in *In*.
- An error occurs if there are any underbars (‘_’) between the minus signs (‘–’) or plus sign (‘+’) and the number at the beginning of *In*.
- If the content of *In* exceeds the precision of the data type of *Out*, the value is rounded.
- If the content of *In* is closer to 0 than the minimum value of the data type of *Out*, the value of *Out* will be 0.
- If the content of *In* exceeds the valid range of *Out*, *Out* will be positive infinity for a positive number or negative infinity for a negative number.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The text string in *In* does not express a number.
  - The text string in *In* has a decimal point but not a fractional part.

TO_** (Integer Conversion Group)

These instructions convert integers, bit strings, real numbers, and text strings to integers.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TO_**	Integer Conversion Group	FUN	 "**" must be an integer data type.	Out:=TO_** (In); "**" must be an integer data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*1	---	*2
Out	Conversion result	Output	Conversion result	*1	---	---

*1 The valid ranges depend on the data types of *In* and *Out*.

*2 If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					OK
Out						OK	OK	OK	OK	OK	OK	OK	OK							

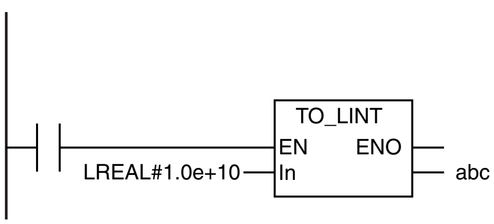
Function

These instructions convert the integer, bit string, real number, or text string in *In* to an integer.

The name of the instruction is determined by the data type of conversion result *Out*. For example, if *Out* is the LINT data type, the instruction is TO_LINT.

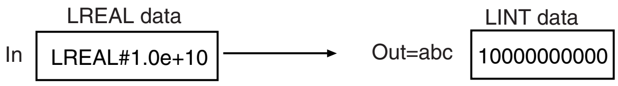
The following example for the TO_LINT instruction is for when *In* is LREAL#1.0e+10.

LD



ST

abc:=TO_LINT(LREAL#1.0e+10);



- Conversion is performed to within the effective digits of the data type of *In.* If *In* is a real number, the fractional part is rounded off to the closest integer. The following table shows how values are rounded.

Value of fractional part	Treatment	Examples
Less than 0.5	The fractional part is truncated.	1.49 → 1
0.5	If the ones digit is an even number, the fractional part is truncated. If it is an odd number, the value is rounded up.	1.50 → 2 2.50 → 2
Greater than 0.5	The fractional part is rounded up.	1.51 → 2

The valid ranges for *In* and *Out* depend on their data types. Refer to the descriptions of the functions of the following instructions for the valid ranges: **_TO_*** (Integer-to-Integer Conversion Group) (page 2-262), **_TO_*** (Bit String-to-Integer Conversion Group) (page 2-270), and **_TO_*** (Real Number-to-Integer Conversion Group) (page 2-276).

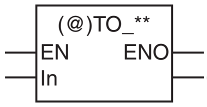
For detailed specifications when *In* is STRING data, refer to Function for the STRING_TO_** (Text String-to-Integer Conversion Group) instructions (page 2-299).

Precautions for Correct Use

- Always use the correct instruction name for the data type of *Out*.
- If the data type of *In* is for a bit string and the sizes of the data types of *In* and *Out* are different, the following processing is performed.
  - If the data size of *Out* is larger than the data size of *In*, the upper digits of *Out* will contain 0.
  - If the data size of *Out* is smaller than the data size of *In*, the upper digits are truncated in *Out*.
- Observe the following precautions if *In* is STRING data.
  - If the first character in *In* is a minus sign (–) or a plus sign (+), it is processed as the sign.
  - Except for a minus sign (–) or a plus sign (+) at the beginning, *In* must consist of consecutive ‘0’ to ‘9’ characters. Underbars (‘_’) and blank characters before or after the ‘–’ or ‘+’ are allowed in the text string.
- If the conversion result exceeds the valid range of *Out*, *Out* will contain an undefined value. Always make sure that the value of *In* is within the valid range so that the conversion result will not exceed the valid range of *Out*.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - *In* is STRING data, but the text sting in *In* does not express a number.

TO_** (Bit String Conversion Group)

These instructions convert integers, bit strings, real numbers, and text strings to bit strings.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TO_**	Bit String Conversion Group	FUN	 "**" must be a bit string data type.	Out:=TO_**(In); "**" must be a bit string data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*1	---	*2
Out	Conversion result	Output	Conversion result	*1	---	---

*1 The valid ranges depend on the data types of *In* and *Out*.

*2 If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					OK
Out		OK	OK	OK	OK															

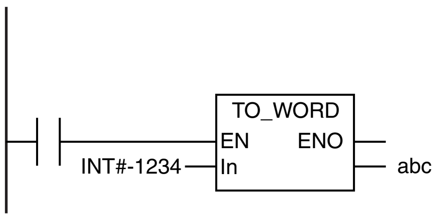
Function

These instructions convert the integer, bit string, real number, or text string in *In* to a bit string.

The name of the instruction is determined by the data type of conversion result *Out*. For example, if *Out* is the WORD data type, the instruction is TO_WORD.

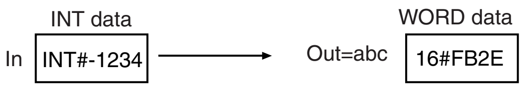
The following example for the TO_WORD instruction is for when *In* is INT#–1234.

LD



ST

abc:=TO_WORD(INT#-1234);



The valid ranges for *In* and *Out* depend on their data types. Refer to the descriptions of the functions of the following instructions for the valid ranges: **_TO_*** (Integer-to-Bit String Conversion Group) (page 2-265), **_TO_*** (Bit String-to-Bit String Conversion Group) (page 2-272), and **_TO_*** (Real Number-to-Bit String Conversion Group) (page 2-279).

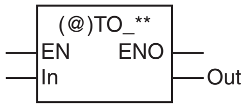
For detailed specifications when *In* is STRING data, refer to Function for the STRING_TO_** (Text String-to-Bit String Conversion Group) instructions (page 2-301).

Precautions for Correct Use

- Always use the correct instruction name for the data type of *Out*.
- If the conversion result exceeds the valid range of *Out*, *Out* will contain an undefined value. Always make sure that the value of *In* is within the valid range so that the conversion result will not exceed the valid range of *Out*.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - *In* is STRING data, but the text sting in *In* does not express a number.

TO_** (Real Number Conversion Group)

These instructions convert integers, bit strings, real numbers, and text strings to real numbers.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TO_**	Real Number Conversion Group	FUN	 "**" must be a real number data type.	Out:=TO_**(In); "**" must be a real number data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	*1, *2	---	*3
Out	Conversion result	Output	Conversion result	*1	---	---

*1 The valid ranges depend on the data types of *In* and *Out*.

*2 For STRING data, the valid range is 311 bytes max. (310 single-byte alphanumeric characters plus the final NULL character).

*3 If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					OK
Out														OK	OK					

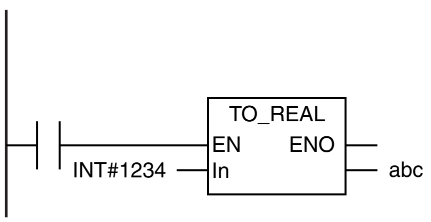
Function

These instructions convert the integer, bit string, real number, or text string in *In* to a real number.

The name of the instruction is determined by the data type of conversion result *Out*. For example, if *Out* is the REAL data type, the instruction is TO_REAL. If the value of *In* is positive or negative infinity, the value of *Out* is positive or negative infinity.

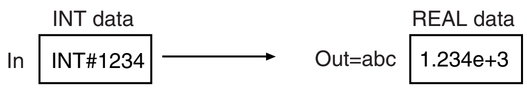
The following example for the TO_REAL instruction is for when *In* is INT#1234.

LD



ST

```
abc:=TO_REAL(INT#1234);
```



The valid ranges for *In* and *Out* depend on their data types. Refer to the descriptions of the functions of the following instructions for the valid ranges: **_TO_*** (Integer-to-Real Number Conversion Group) (page 2-268), **_TO_*** (Bit String-to-Real Number Conversion Group) (page 2-274), and **_TO_*** (Real Number-to-Real Number Conversion Group) (page 2-281).

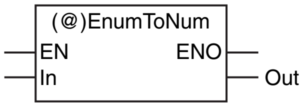
For detailed specifications when *In* is STRING data, refer to Function for the STRING_TO_** (Text String-to-Real Number Conversion Group) instructions (page 2-303).

Precautions for Correct Use

- Always use the correct instruction name for the data type of *Out*.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - *In* is STRING data, but the text sting in *In* does not express a number.

EnumToNum

The EnumToNum instruction converts enumeration data to DINT data.

Instruction	Name	FB/FUN	Graphic expression	ST expression
EnumToNum	Enumeration-to-Integer	FUN		Out:=EnumToNum(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	---	---	0
Out	Conversion result	Output	Conversion result	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	Enumeration																			
Out												OK								

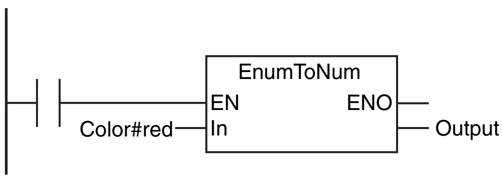
Function

The EnumToNum instruction converts the value of data to convert *In*, which is an enumeration, to a DINT value and outputs the value to conversion result *Out*.

Use this instruction, for example, to monitor the value of an enumerated variable on an HMI or other display device that does not handle enumerated variables.

The following example shows how to convert enumerator *red* of the enumeration *Color* to a value and output that value to DINT variable *Output*. If the value of enumerator red is 0, *Output* will be DINT#0.

LD



ST

```
Output:=EnumToNum(Color#red);
```

Precautions for Correct Use

Version Information

A CPU Unit with unit version 1.02 or later and Sysmac Studio version 1.03 or higher are required to use this instruction.

Sample Programming

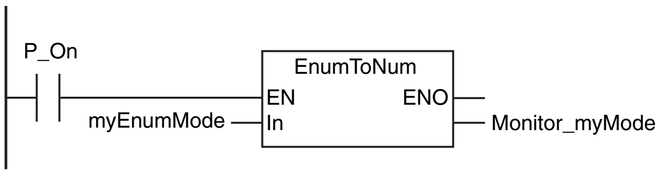
In this sample, the operating mode of the user program is defined with enumerated data type EnumMode. To monitor the operating mode on the HMI, the value of variable *myEnumMode* (an enumeration with a data type of EnumMode) is converted and the converted value is output to DINT variable *Monitor_myMode*. For example, if the value of *myEnumMode* is mode2, the value of *Monitor_myMode* will be 2.

● Data Type Definition

Name	Enumeration value	Comment
EnumMode	---	Enumerated data type
mode0	0	Member
mode1	1	Member
mode2	2	Member

LD

Name	Data type	Default	Comment
myEnumMode	EnumMode	mode0	Value of mode in enumerated data type
Monitor_myMode	DINT	0	Monitored mode value



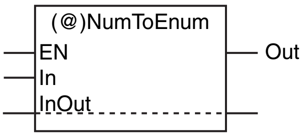
ST

Name	Data type	Default	Comment
myEnumMode	EnumMode	mode0	Value of mode in enumerated data type
Monitor_myMode	DINT	0	Monitored mode value

```
Monitor_myMode:=EnumToNum(myEnumMode);
```

NumToEnum

The NumToEnum instruction converts DINT data to enumeration data.

Instruction	Name	FB/FUN	Graphic expression	ST expression
NumToEnum	Integer-to-Enumeration	FUN		NumToEnum(In,InOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	Depends on data type.	---	0
InOut	Conversion result	In-out	Conversion result	---	---	---
Out	Return value	Output	TRUE: Instruction was executed normally. FALSE: Instruction was not executed or an error occurred.	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In												OK								
InOut	Enumeration																			
Out	OK																			

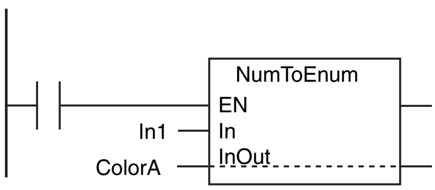
Function

The NumToEnum instruction converts the value of data to convert *In*, which is DINT data, to an enumeration value and outputs that value to conversion result *InOut*.

Use this instruction, for example, to change the value of an enumerated variable from an HMI or other display device that does not handle enumerated variables.

The following example shows how to convert the value of DINT variable *In1* and output the results to variable *ColorA*, which has an enumerated data type of *Color*. If *green* is the enumerator that corresponds to an enumeration value of 1 for *Color* and the value of *In1* is 1, the value of *ColorA* will be *green*.

LD



ST

```
NumToEnum(In1,ColorA);
```

Additional Information

If you use this instruction in a ladder diagram, you can use *Out* to see if the value of *In* is within the range of values for *InOut*.

Precautions for Correct Use

An error occurs if the value of *In* is not within the range of values for *InOut*. *Out* will be FALSE, and the value of *InOut* will not change.

Version Information

A CPU Unit with unit version 1.02 or later and Sysmac Studio version 1.03 or higher are required to use this instruction.

Sample Programming

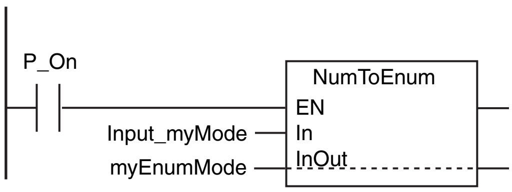
In this sample, the operating mode of the user program is defined with enumerated data type EnumMode. To change the operating mode from an HMI, the value of *Input_myMode*, which is a DINT variable, is written. In the user program, the value of *Input_myMode* is converted and the converted value is output to variable *myEnumMode* (an enumeration with a data type of EnumMode). For example, if the value of *Input_myMode* is 1, the value of *myEnumMode* will be mode1.

● Data Type Definition

Name	Enumeration value	Comment
EnumMode	---	Enumerated data type
mode0	0	Member
mode1	1	Member
mode2	2	Member

LD

Name	Data type	Default	Comment
myEnumMode	EnumMode	mode0	Value of mode in enumerated data type
Input_myMode	DINT	0	Value of mode to which to change



ST

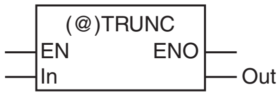
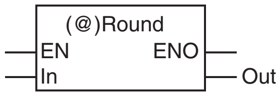
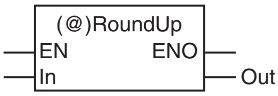
Name	Data type	Default	Comment
myEnumMode	EnumMode	mode0	Value of mode in enumerated data type
Input_myMode	DINT	0	Value of mode to which to change

```
NumToEnum(Input_myMode, myEnumMode);
```

TRUNC, Round, and RoundUp

These instructions change real numbers to integers.

TRUNC:	Truncates the number at the first decimal digit.
Round:	Rounds the number at the first decimal digit.
RoundUp:	Rounds up the number at the first decimal digit.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TRUNC	Truncate	FUN		Out:=TRUNC(In);
Round	Round Off Real Number	FUN		Out:=Round(In);
RoundUp	Round Up Real Number	FUN		Out:=RoundUp(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	Depends on data type.	---	*
Out	Conversion result	Output	Conversion result	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK	OK					
Out												OK	OK							

Function

These instructions change the real number in *In* to an integer by eliminating the fractional part.

● TRUNC

The TRUNC instruction truncates the number at the first decimal digit.

● Round

The Round instruction rounds the number at the first decimal digit. The following table shows how values are rounded.

Value of fractional part	Treatment	Examples
Less than 0.5	The fractional part is truncated.	1.49 → 1 –1.49 → –1
0.5	If the ones digit is an even number, the fractional part is truncated. If it is an odd number, the value is rounded up.	1.50 → 2 2.50 → 2 –1.50 → –2 –2.50 → –2
Greater than 0.5	The fractional part is rounded up.	1.51 → 2 –1.51 → –2

● RoundUp

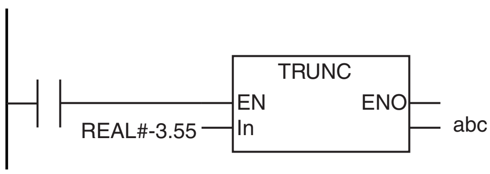
The RoundUp instruction rounds up the number at the first decimal digit.

The differences in these three instructions are shown by the following examples.

Input value	Output value		
	TRUNC	Round	RoundUp
REAL#1.6	DINT#1	DINT#2	DINT#2
REAL#1.5	DINT#1	DINT#2	DINT#2
REAL#1.5	DINT#1	DINT#1	DINT#2
REAL#2.5	DINT#2	DINT#2	DINT#3
REAL#-1.6	DINT#-1	DINT#-2	DINT#-2
REAL#-1.5	DINT#-1	DINT#-2	DINT#-2
REAL#-1.4	DINT#-1	DINT#-1	DINT#-2
REAL#-2.5	DINT#-2	DINT#-2	DINT#-3

The following example for the TRUNC instruction is for when *In* is REAL#–3.55. The value of variable *abc* will be DINT#–3.

LD



ST

```
abc:=TRUNC(REAL#-3.55);
```

The TRUNC instruction truncates the number at the first decimal digit.
The value of *In* is REAL#–3.55, so the value of *abc* will be DINT#–3.

In $\boxed{\text{REAL\#-3.55}}$ —Truncated at decimal point.→ Out=abc $\boxed{\text{DINT\#-3}}$

Additional Information

If the data type of *In* is REAL, the data type of *Out* is DINT. If the data type of *In* is LREAL, the data type of *Out* is LINT.

Precautions for Correct Use

If the conversion result exceeds the valid range of *Out*, *Out* will contain an undefined value. Always make sure that the value of *In* is within the valid range so that the conversion result will not exceed the valid range of *Out*.

Bit String Processing Instructions

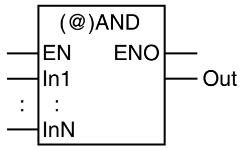
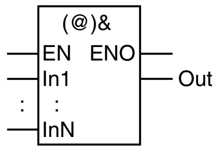
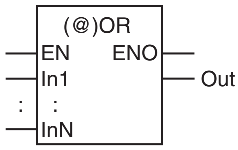
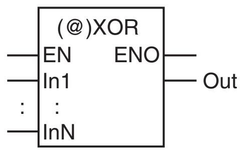
AND (&), OR, and XOR

These instructions perform processing on Boolean variables or individual bits in bit stings.

AND (&): Logical AND

OR: Logical OR

XOR: Logical Exclusive OR

Instruction	Name	FB/FUN	Graphic expression	ST expression
AND (&)	Logical AND	FUN		Out:=In1 AND ··AND InN; Out:=In1 & ··& InN;
OR	Logical OR	FUN		Out:=In1 OR ··OR InN;
XOR	Logical Exclusive OR	FUN		Out:=In1 XOR ··XOR InN;

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1 to InN	Data to process	Input	Data to process, where N is 2 to 5	Depends on data type.	---	0*
Out	Processing result	Output	Processing result	Depends on data type.	---	---

* If you omit the input parameter that connects to *InN*, the default value is not applied, and a building error will occur. For example, if N is 3 and the input parameters that connect to *In1* and *In2* are omitted, the default values are applied, but if the input parameter that connects to *In3* is omitted, a building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN	OK	OK	OK	OK	OK															
Out	Must be same data type as *In1* to *InN*																			

Function

These instructions perform processing on Boolean variables or corresponding bits in bit strings. The data to process is in *In1* to *InN*. *In1* to *InN* and *Out* must be the same data types.

If there are more than two data to process, processing is performed with the following procedure.

1. Processing is performed for *In1* and *In2*.
2. Processing is performed for the results of step 1 and *In3*.
3. Processing is performed for the results of step 2 and *In4*.

⋮ ⋮

The relationships between input and output variables are given in the following tables.

● AND (&)

If both bits are TRUE, then the processing result is TRUE. Otherwise, the processing result is FALSE.

In1 bit	*In2* bit	*Out* bit
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

● OR

If both bits are FALSE, then the processing result is FALSE. Otherwise, the processing result is TRUE.

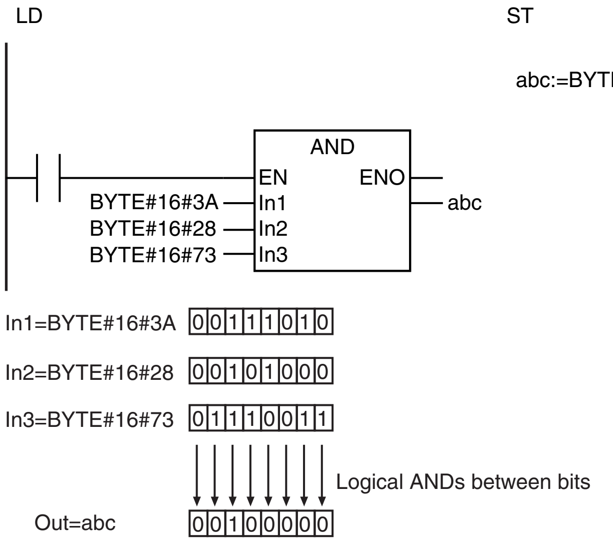
In1 bit	*In2* bit	*Out* bit
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

● XOR

If both bits are the same, then the processing result is FALSE. If one bit is TRUE and the other is FALSE, then the processing result is TRUE.

In1 bit	*In2* bit	*Out* bit
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	FALSE

The following example shows the AND instruction when *In1* is BYTE#16#3A, *In2* is BYTE#16#28 and *In3* is BYTE#16#73.



The functions of the AND instruction and the & instruction are exactly the same. Use the form that is easier to use.

Additional Information

In ST, there is no limit to the number of input variables if you use the following notation.

Out:=In1 AND In2 AND In3 AND In4 AND In5 AND In6 ···

Out:=In1 & In2 & In3 & In4 & In5 & In6 ···

Out:=In1 OR In2 OR In3 OR In4 OR In5 OR In6 ···

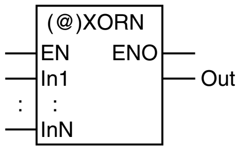
Out:=In1 XOR In2 XOR In3 XOR In4 XOR In5 XOR In6 ···

Precautions for Correct Use

The data types of *In1* to *InN* and *Out* must all be the same. Otherwise, a building error will occur.

XORN

The XORN instruction performs a logical exclusive NOR operation on Boolean variables or individual bits in bit stings.

Instruction	Name	FB/FUN	Graphic expression	ST expression
XORN	Logical Exclusive NOR	FUN		Out:=In1 XOR NOT ·· XOR NOT InN;

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1 to InN	Data to process	Input	Data to process, where N is 2 to 5	Depends on data type.	---	0*
Out	Processing result	Output	Processing result	Depends on data type.	---	---

* If you omit the input parameter that connects to *InN*, the default value is not applied, and a building error will occur. For example, if N is 3 and the input parameters that connect to *In1* and *In2* are omitted, the default values are applied, but if the input parameter that connects to *In3* is omitted, a building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN	OK	OK	OK	OK	OK															
Out	Must be same data type as *In1* to *InN*																			

Function

The XORN instruction performs processing on Boolean variables or corresponding bits in bit strings. The data to process is in *In1* to *InN. In1* to *InN* and *Out* must be the same data types.

If there are more than two data to process, processing is performed with the following procedure.

1. Processing is performed for *In1* and *In2*.
2. Processing is performed for the results of step 1 and *In3*.
3. Processing is performed for the results of step 2 and *In4*.

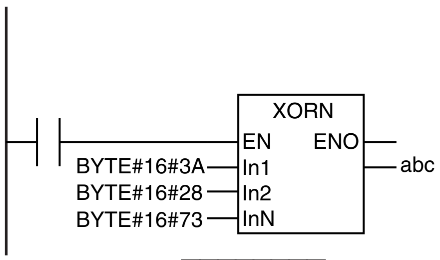
⋮ ⋮

The relationships between input and output variables are given in the following table. If both values are the same, then the processing result is TRUE. Otherwise, the processing result is FALSE.

***In1* bit**	***In2* bit**	***Out* bit**
FALSE	FALSE	TRUE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

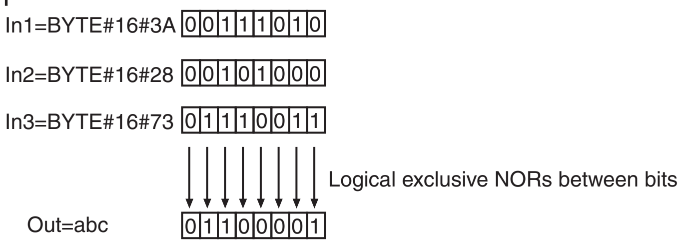
The following example is for when *In1* is BYTE#16#3A, *In2* is BYTE#16#28, and *In3* is BYTE#16#73.

LD



ST

```
abc:=BYTE#16#3A XOR NOT BYTE#16#28
        XOR NOT BYTE#16#73;
```

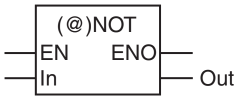


Precautions for Correct Use

The data types of *In1* to *InN* and *Out* must all be the same. Otherwise, a building error will occur.

NOT

The NOT instruction reverses the value of a Boolean variable or the individual bits in a bit string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
NOT	Bit Reversal	FUN		Out:=NOT In;

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to process	Input	Data to process	Depends on data type.	---	*
Out	Processing result	Output	Processing result	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

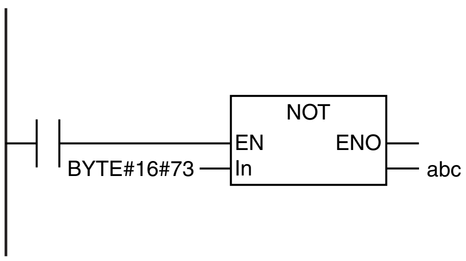
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	OK	OK	OK	OK	OK															
Out	Must be same data type as *In*																			

Function

The NOT instruction reverses the value of a Boolean variable or the values of individual bits in a bit string. The data to process is in *In*. *In* and processing result *Out* must have the same number of bits, i.e., they must be the same data type.

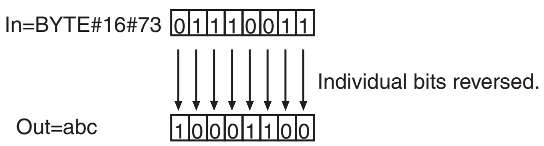
The following example is for when *In* is BYTE#16#73.

LD



ST

abc:=NOT(BYTE#16#73);



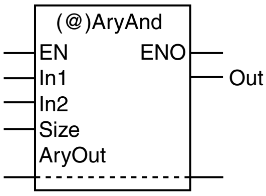
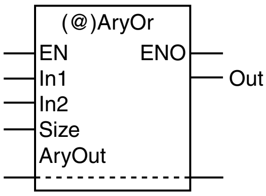
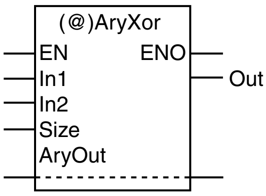
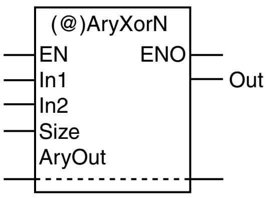
Precautions for Correct Use

The data types of *In* and *Out* must be the same. Otherwise, a building error will occur.

AryAnd, AryOr, AryXor, and AryXorN

These instructions process Boolean variables or individual bits in bit stings between arrays.

AryAnd: Logical AND
AryOr: Logical OR
AryXor: Logical Exclusive OR
AryXorN: Logical Exclusive NOR

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryAnd	Array Logical AND	FUN		AryAnd(In1, In2, Size, AryOut);
AryOr	Array Logical OR	FUN		AryOr(In1, In2, Size, AryOut);
AryXor	Array Logical Exclusive OR	FUN		AryXor(In1, In2, Size, AryOut);
AryXorN	Array Logical Exclusive NOR	FUN		AryXorN(In1, In2, Size, AryOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1[] and In2[] (arrays)	Array to process	Input	Array to process	Depends on data type.	---	*
Size	Number of elements		Number of elements to process			1
AryOut[] (array)	Processing results array	In-out	Processing results array	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[] (array)	OK	OK	OK	OK	OK															
In2[] (array)	Must be same data type as *In1[]*																			
Size							OK													
AryOut[] (array)	Must be same data type as *In1[]*																			
Out	OK																			

Function

These instructions process *Size* elements from the beginning of arrays to process *In1[]* and *In2[]*. Processing is performed for corresponding bits of corresponding elements. The processing results are stored in corresponding elements of *AryOut[]*. *In1[]* to *In2[]* and *AryOut[]* must be the same data types.

The relationships between input and output variables are given in the following tables.

● AryAnd

If both bits are TRUE, then the processing result is TRUE. Otherwise, the processing result is FALSE.

Bit of element in *In1[]*	Bit of element in *In2[]*	Bit of *AryOut[]*
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

● AryOr

If both bits are FALSE, then the processing result is FALSE. Otherwise, the processing result is TRUE.

Bit of element in *In1[]*	Bit of element in *In2[]*	Bit of *AryOut[]*
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

- **AryXor**

If both bits are the same, then the processing result is FALSE. If one bit is TRUE and the other is FALSE, then the processing result is TRUE.

Bit of element in *In1[]*	Bit of element in *In2[]*	Bit of *AryOut[]*
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	FALSE

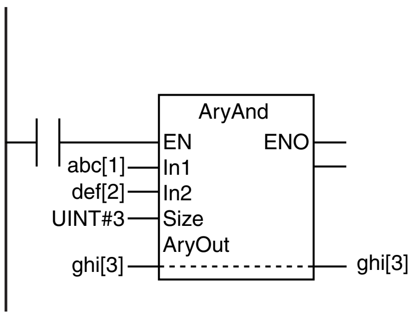
- **AryXorN**

If both bits are the same, then the processing result is TRUE. If one bit is TRUE and the other is FALSE, then the processing result is FALSE.

Bit of element in *In1[]*	Bit of element in *In2[]*	Bit of *AryOut[]*
FALSE	FALSE	TRUE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

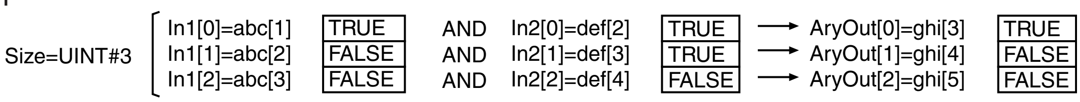
The following example shows the AryAnd instruction when *Size* is UINT#3.

LD



ST

```
AryAnd(abc[1], def[2], UINT#3, ghi[3]);
```



Precautions for Correct Use

- The data types of *In1[]*, *In2[]*, and *AryOut[]* must be the same. If they are different, a building error will occur.
- Use an *AryOut[]* array that has at least as many elements as the value of *Size.*
- The values in *AryOut[]* do not change if the value of *Size* is 0.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *AryOut[]* will not change.
  - The value of *Size* exceeds the number of elements in *In1[], In2[],* or *AryOut[].*

Selection Instructions

SEL

The SEL instruction selects one of two selections.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SEL	Binary Selection	FUN	(@)SEL EN ENO G Out In0 In1	Out:=SEL(G, In0, In1);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
G	Gate	Input	FALSE: Selects *In0.* TRUE: Selects *In1.*	Depends on data type.	---	FALSE
In0 and In1	Selections	Input	Selections	Depends on data type.	---	*
Out	Selection result	Output	Selection result	Depends on data type.	---	---

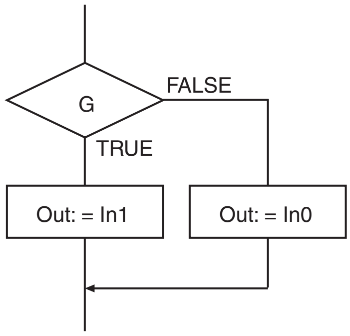
* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
G	OK																			
In0 and In1	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	Enumerations can also be specified.*																			
Out	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	Enumerations can also be specified.*																			

* A CPU Unit with unit version 1.02 or later and Sysmac Studio version 1.03 or higher are required to specify enumerations.

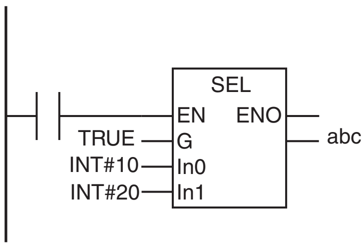
Function

The SEL instruction selects one of two selections, *In0* and *In1*. Gate *G* specifies which of *In0* and *In1* to select. If *G* is FALSE, *In0* is assigned to *Out*. If *G* is TRUE, *In1* is assigned to *Out*.



The following example is for when *In0* is INT#10, *In1* is INT#20, and *G* is TRUE. The value of variable *abc* will be INT#20.

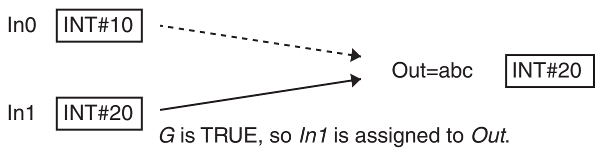
LD



ST

```
abc:=SEL(TRUE, INT#10, INT#20);
```

The SEL instruction selects *In0* or *In1*.
G is TRUE, so *In1* (INT#20) is selected and assigned to *abc*.



G is TRUE, so *In1* is assigned to *Out*.

Additional Information

Version Information

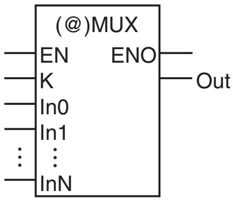
With a CPU Unit with unit version 1.02 or later and Sysmac Studio version 1.03 or higher, the MUX instruction (page 2-334) can also be used.

Precautions for Correct Use

- *In0, In1,* and *Out* may be different data types, but observe the following precautions.
  - Set the valid range of *Out* to include the valid ranges of *In0* and *In1*.
  - *In0, In1,* and *Out* cannot be different varieties of data types (such as a bit string and an integer, or an integer and a text string).

MUX

The MUX instruction selects one of two to five selections.

Instruction	Name	FB/FUN	Graphic expression	ST expression
MUX	Multiplexer	FUN		Out:=MUX(K, In0, In1, ···, InN);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
K	Selector	Input	0: Selects *In0.* 1: Selects *In1.* 2: Selects *In2.* 3: Selects *In3.* 4: Selects *In4.*	0 to N	---	*1
In0 to InN	Selections	Input	Selections N is 1 to 4.*2	Depends on data type.	---	0*3
Out	Selection result	Output	Selection result	Depends on data type.	---	---

*1 If you omit an input parameter, the default value is not applied. A building error will occur.

*2 With a CPU Unit with unit version 1.01 or earlier and Sysmac Studio version 1.02 or lower, N is 2 to 4.

*3 If you omit the input parameter that connects to *InN*, the default value is not applied, and a building error will occur. For example, if N is 2 and the input parameters that connect to *In0* and *In1* are omitted, the default values are applied, but if the input parameter that connects to *In2* is omitted, a building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
K						OK*1			OK*1											
In0 to InN	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	Enumerations can also be specified.*2																			
Out	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	Enumerations can also be specified.*2																			

*1 With a CPU Unit with unit version 1.02 or later and Sysmac Studio version 1.03 or higher, use a ULINT variable. With a CPU Unit with unit version 1.01 or earlier and Sysmac Studio version 1.02 or lower, use a USINT variable.

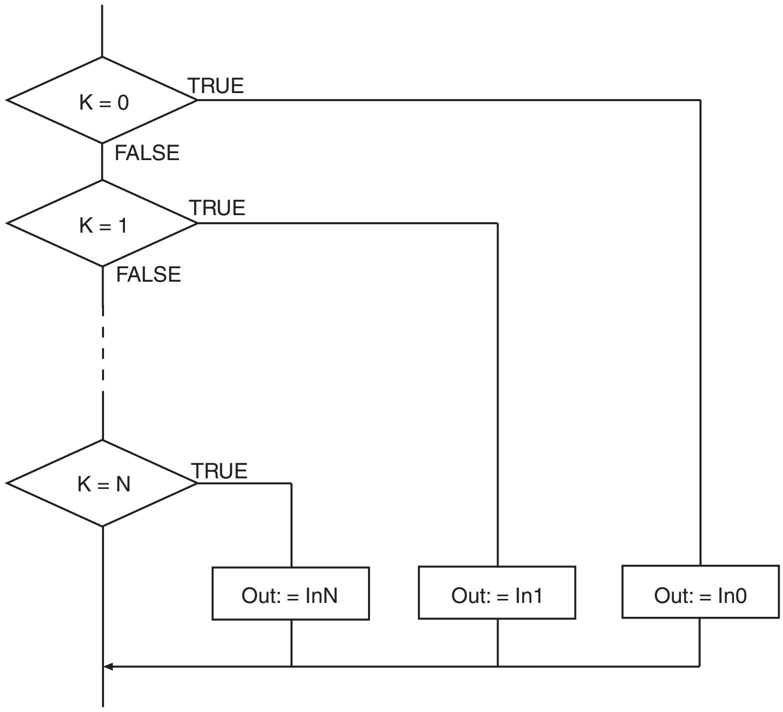
*2 A CPU Unit with unit version 1.02 or later and Sysmac Studio version 1.03 or higher are required to specify enumerations.

Function

The MUX instruction selects one of two to five selections, *In0* to *InN*.

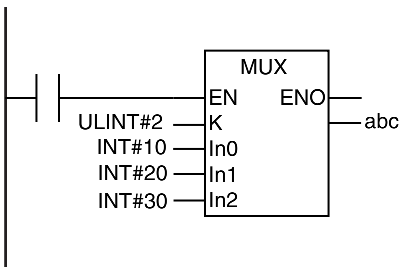
Selector *K* specifies which of *In0* to *InN* to select.

The value of one of the input variables is assigned to *Out* according to the value of *K. In0* is assigned if *K* is 0, *In1* is assigned if *K* is 1, *etc.*



The following example is for when In0 is INT#10, *In1* is INT#20, *In2* is INT#30, and K is ULINT#2. The value of variable *abc* will be INT#30.

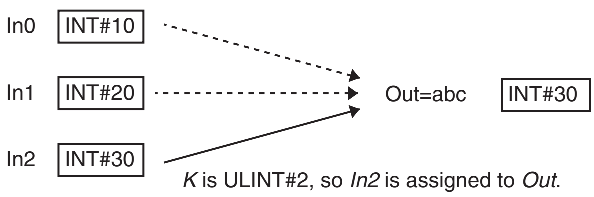
LD



ST

abc:=MUX(ULINT#2, INT#10, INT#20, INT#30);

The MUX instruction selects from among *In0* to *InN*.
K is ULINT#2, so *In2* (INT#30) is selected and assigned to *abc*.



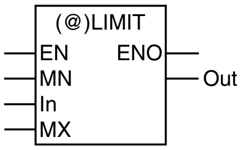
K is ULINT#2, so *In2* is assigned to *Out*.

Precautions for Correct Use

- *In0* to *InN* and *Out* may be different data types, but observe the following precautions.
  - Set the valid range of *Out* to include the valid ranges of *In0* to *InN*.
  - *In0* to *InN* and *Out* cannot be different varieties of data types (such as a bit string and an integer, or an integer and a text string).
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of *K* is outside the valid range (i.e., less than 0 or greater than N).

LIMIT

The LIMIT instruction limits the value of the input variable to the specified minimum and maximum values.

Instruction	Name	FB/FUN	Graphic expression	ST expression
LIMIT	Limiter	FUN		Out:=LIMIT(MN, In, MX);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
MN	Minimum value	Input	Minimum value of limiter	Depends on data type.	---	*
In	Data to limit		Data to limit			
MX	Maximum value		Maximum value of limiter			
Out	Processing result	Output	Processing result	Depends on data type.	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
MN						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
In						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
MX						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Out						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					

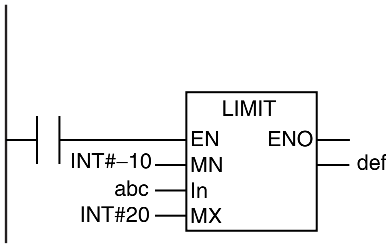
Function

The LIMIT instruction limits the value of data to limit *In* according to the maximum value, *MX*, and the minimum value, *MN*. The value of processing result *Out* is as shown below.

Value of *In*	Value of *Out*
In < MN	MN
MN ≤ In ≤ MX	In
MX < In	MX

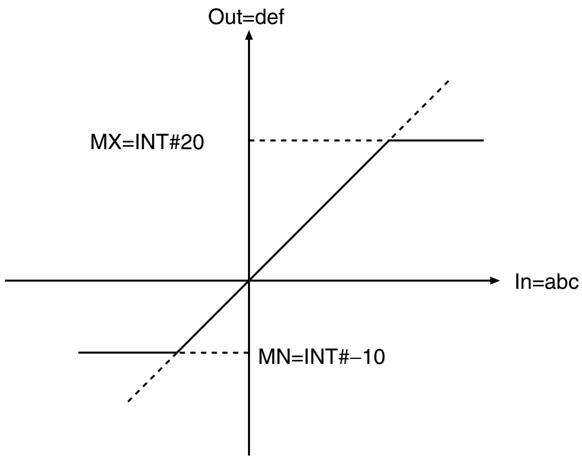
The following example is for when *MN* is INT#–10 and *MX* is INT#20.

LD



ST

```
def:=LIMIT(INT#–10, abc, INT#20);
```

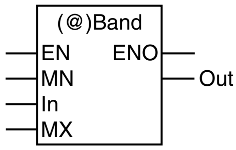


Precautions for Correct Use

- *In, MN, MX,* and *Out* may be different data types, but observe the following precautions.
  - Set the valid range of *Out* to include the valid ranges of *In, MN*, and *MX*.
  - Do not combine signed integers (SINT, INT, DINT, and LINT) together with unsigned integers (USINT, UINT, UDINT, and ULINT) for *In*, *MN*, and *MX*.
- An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.
  - The value of *MX* is smaller than the value of *MN.*

Band

The Band instruction performs deadband control.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Band	Deadband Control	FUN		Out:=Band(MN, In, MX);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
MN	Minimum value	Input	Minimum value of deadband	Depends on data type.	---	*
In	Data to control		Data to control			
MX	Maximum value		Maximum value of deadband			
Out	Processing result	Output	Processing result	Depends on data type.	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
MN										OK	OK	OK	OK	OK	OK					
In										OK	OK	OK	OK	OK	OK					
MX										OK	OK	OK	OK	OK	OK					
Out										OK	OK	OK	OK	OK	OK					

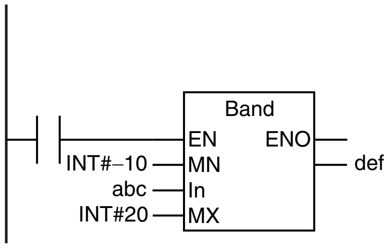
Function

The Band instruction controls the value of data to control *In* according to the maximum value, *MX*, and the minimum value, *MN*. The value of processing result *Out* is as shown below.

Value of *In*	Value of *Out*
In < MN	In – MN
MN ≤ In ≤ MX	0
MX < In	In – MX

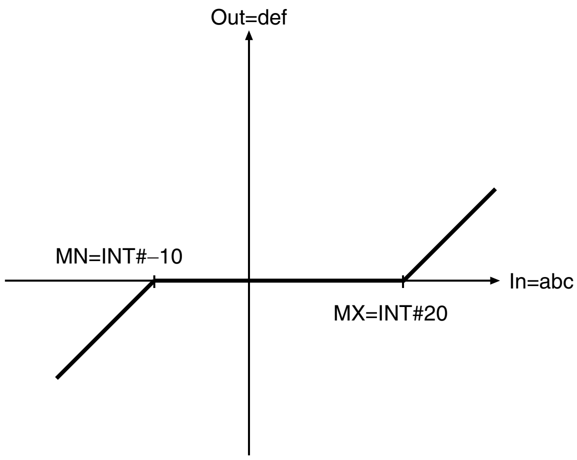
The following example is for when *MN* is INT#–10 and *MX* is INT#20.

LD



ST

def:=Band(INT#–10, abc, INT#20);



Precautions for Correct Use

- *In, MN, MX,* and *Out* may be different data types, but observe the following precaution.
  - Set the valid range of *Out* to include the valid ranges of *In, MN*, and *MX*.
- If the value of *In* is nonnumeric data, the value of *Out* is nonnumeric data.
- If the value of *In*, *MN*, or *MX* is positive infinity or negative infinity, the value of *Out* is as shown below.

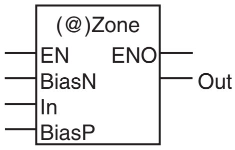
Value of *In*	Value of *MN*	Value of *MX*	Value of *Out*
+∞	+∞	+∞	0
		–∞	Error
	–∞	+∞	0
		–∞	+∞

Value of *In*	Value of *MN*	Value of *MX*	Value of *Out*
–∞	+∞	+∞	–∞
		–∞	Error
	–∞	+∞	0
		–∞	0

- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of *MX* is smaller than the value of *MN.*
  - Either *MX* or *MN* contains nonnumeric data.
  - The processing result exceeds the valid range of *Out*.

Zone

The Zone instruction adds a bias value to the input value.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Zone	Dead Zone Control	FUN		Out:=Zone(BiasN, In, BiasP);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
BiasN	Negative bias	Input	Negative bias	Depends on data type.	---	*
In	Data to control		Data to control			
BiasP	Positive bias		Positive bias			
Out	Processing result	Output	Processing result	Depends on data type.	---	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
BiasN										OK	OK	OK	OK	OK	OK					
In										OK	OK	OK	OK	OK	OK					
BiasP										OK	OK	OK	OK	OK	OK					
Out										OK	OK	OK	OK	OK	OK					

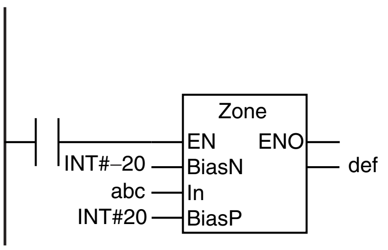
Function

The Zone instruction controls the value of data to control *In* according to the positive bias, *BiasP*, and the negative bias, *BiasN*. The value of processing result *Out* is as shown below.

Value of *In*	Value of *Out*
In < 0	In + BiasN
In = 0	0
0 < In	In + BiasP

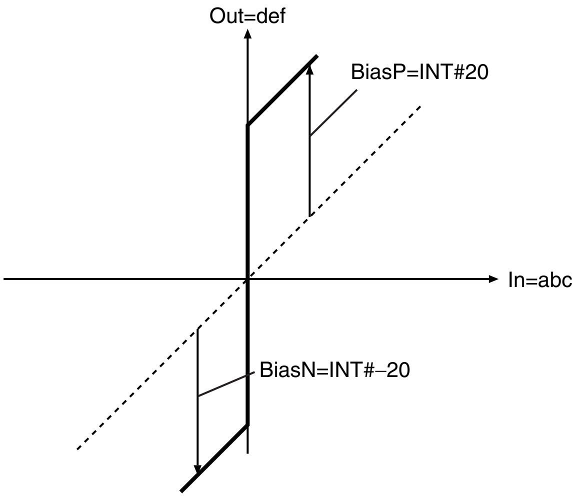
The following example is for when *BiasP* is INT#20 and *BiasN* is INT#–20.

LD



ST

```
def:=Zone(INT#–20, abc, INT#20);
```



Precautions for Correct Use

- *In, BiasP, BiasN,* and *Out* may be different data types, but observe the following precaution.
  - Set the valid range of *Out* to include the valid ranges of *In, BiasP*, and *BiasN*.
- If the value of *In* is nonnumeric data, the value of *Out* is nonnumeric data.
- If the value of *In*, *BiasP*, or *BiasN* is positive infinity or negative infinity, the value of *Out* is as shown below.

Value of *In*	Value of *BiasP*	Value of *BiasN*	Value of *Out*
+∞	+∞	+∞	+∞
		–∞	+∞
	–∞	+∞	Error
		–∞	0

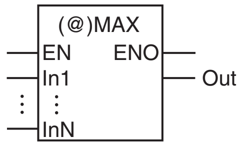
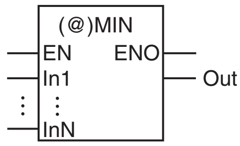
Value of *In*	Value of *BiasP*	Value of *BiasN*	Value of *Out*
−∞	+∞	+∞	0
		−∞	−∞
	−∞	+∞	Error
		−∞	−∞

- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - *BiasP* is less than *BiasN*.
  - Either *BiasP* or *BiasN* contains nonnumeric data.
  - The processing result exceeds the valid range of *Out*.

MAX and MIN

MAX: Finds the largest of two to five values.

MIN: Finds the smallest of two to five values.

Instruction	Name	FB/FUN	Graphic expression	ST expression
MAX	Maximum	FUN		Out:=MAX(In1, In2, ···, InN);
MIN	Minimum	FUN		Out:=MIN(In1, In2, ···, InN);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1 to InN	Data to process	Input	Data to process, where N is 2 to 5	Depends on data type.	---	0*
Out	Search result	Output	Search result	Depends on data type.	---	---

* If you omit the input parameter that connects to *InN*, the default value is not applied, and a building error will occur. For example, if N is 3 and the input parameters that connect to *In1* and *In2* are omitted, the default values are applied, but if the input parameter that connects to *In3* is omitted, a building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Out						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					

Function

- **MAX**

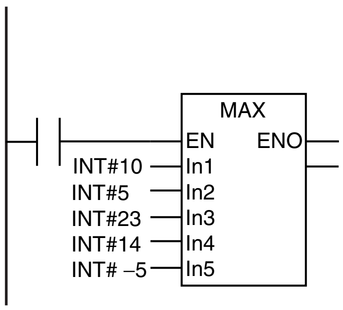
The MAX instruction finds the largest value of two to five data to process, *In1* to *InN*.

- **MIN**

The MIN instruction finds the smallest value of two to five data to process, *In1* to *InN*.

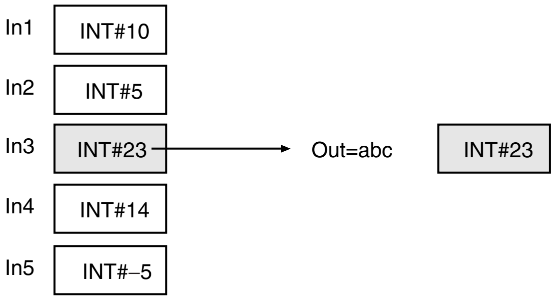
The following example is for the MAX instruction when *In1* is INT#10, *In2* is INT#5, *In3* is INT#23, *In4* is INT#14, and *In5* is INT#–5.

LD



ST

```
abc:=MAX(INT#10, INT#5, INT#23, INT#14, INT#–5);
```



Additional Information

To find the largest or smallest of six or more values, use the AryMax or AryMin instruction (page 2-347).

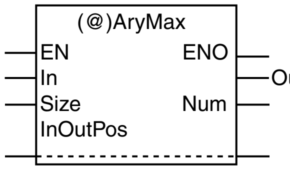
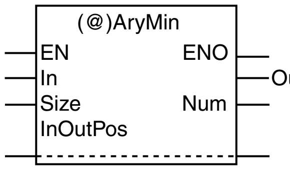
Precautions for Correct Use

- *In1* to *InN* and *Out* may be different data types, but observe the following precaution.
  - Set the valid range of *Out* to include the valid ranges of *In1* to *InN*.
  - Do not combine signed integers (SINT, INT, DINT, and LINT) together with unsigned integers (USINT, UINT, UDINT, and ULINT) for *In1* to *InN*.
- If *In1* to *InN* are real numbers, the desired results may not be achieved due to error.

AryMax and AryMin

AryMax: Finds the elements with the largest value in a one-dimensional array.

AryMin: Finds the elements with the smallest value in a one-dimensional array.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryMax	Array Maximum	FUN		Out:=AryMax(In, Size, InOutPos, Num);
AryMin	Array Minimum	FUN		Out:=AryMin(In, Size, InOutPos, Num);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array to search	Input	Array to search	Depends on data type.	---	*
Size	Number of elements to search		Number of elements in *In[]* to search			1
InOutPos	Found element number	In-out	Array element number where value was found	Depends on data type.	---	---
Out	Search result	Output	Search result	Depends on data type.	---	---
Num	Number found		Number found			

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
																*				
Size							OK													
InOutPos							OK													
Out						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
																*				
Num							OK													

* You can specify TIME, DATE, TOD, DT, and STRING data with CPU Units with unit version 1.01 or later and Sysmac Studio version 1.02 or higher.

Function

These instructions search *Size* elements in array to search *In[]* starting from *In[0]*. The value that is found is assigned to *Out*, the element number where it was found is assigned to *InOutPos*, and the number of times the value was found is assigned to *Num*. If *Num* is greater than 1, the value in *InOutPos* is the number of the lowest element that contains the value that was found.

The relationship between values with data types that are not integers or real numbers are determined as given in the following table.

Data type	Relationship
TIME	The numerically larger value is considered to be larger.
DATE, TOD, or DT	Later dates or times of day are considered to be larger.
STRING	The specifications are the same as for the LTascii, LEascii, GTascii, and GEascii instructions (page 2-104). Refer to the specified page for details.

● AryMax

The AryMax instruction finds the largest value.

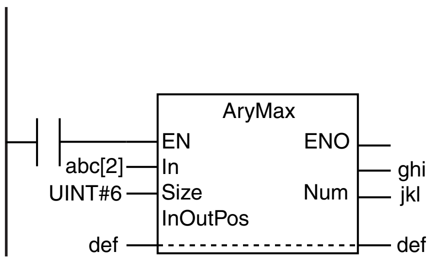
● AryMin

The AryMin instruction finds the smallest value.

The following example shows the AryMax instruction when *Size* is UINT#6.

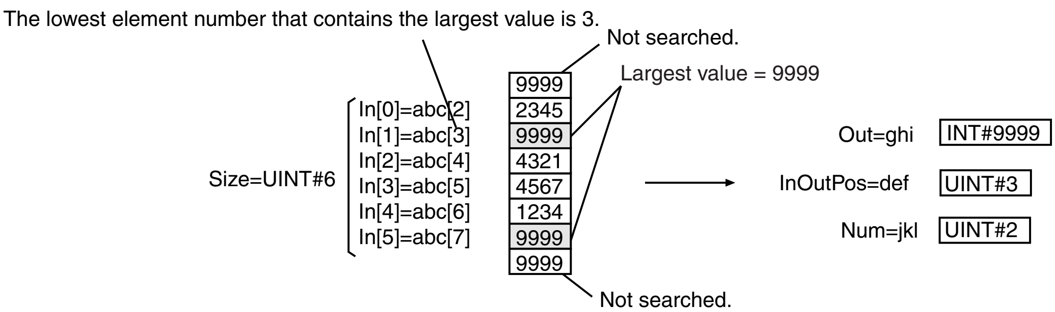
The input parameter that is passed to *In[]* is *abc[2]*, so the search starts from *abc[2]*.

LD



ST

```
ghi:=AryMax(abc[2], UINT#6, def, jkl);
```



Additional Information

When you compare TIME, DT, or TOD data, adjust the data so that the precision of the values is the same. Use the following instructions to adjust the precision of the values: TruncTime (page 2-657), TruncDt (page 2-661), and TruncTod (page 2-665).

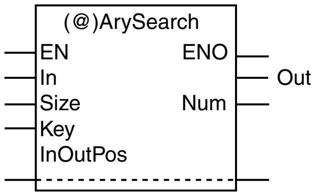
Precautions for Correct Use

- If you use a different data type for *In[]* and *Out*, make sure the valid range of *Out* includes the valid range of *In[]*.

- If *In[]* contains real numbers, the desired results may not be achieved due to error.
- Always used a one-dimensional array for *In[]*.
- If the value of *Size* is 0, the values of *Out* and *Num* are 0. The value of *InOutPos* does not change.
- If *In[]* contains STRING data and the value of *Size* is 0, *Out* is a text string containing only the NULL character.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of *Size* is outside of the valid range.
  - *Size* exceeds the array area of *In[]*.
  - *In[]* is not a one-dimensional array.
  - *In[]* is STRING data and it does not end in a NULL character.

ArySearch

The ArySearch instruction searches for the specified value in a one-dimensional array.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ArySearch	Array Search	FUN		Out:=ArySearch(In, Size, Key, InOutPos, Num);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array to search	Input	Array to search	Depends on data type.	---	*
Size	Number of elements to search		Number of elements in *In[]* to search	1 to 65535		1
Key	Search key		Value to search for	Depends on data type.		---
InOutPos	Found element number	In-out	Array element number where value was found	Depends on data type.	---	---
Out	Search result	Output	Search result	Depends on data type.	---	---
Num	Number found		Number found			

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
																*				
	Arrays of enumerations can also be specified.																			
Size							OK													
Key	Must be same data type as the elements of *In[]*.																			
InOutPos							OK													
Out	OK																			
Num							OK													

* You can specify TIME, DATE, TOD, and DT data with CPU Units with unit version 1.01 or later and Sysmac Studio version 1.02 or higher.

Function

The ArySearch instruction searches *Size* elements of one-dimensional array to search *In[]* for elements with the same value as search key *Key*. The search starts from *In[0]*.

The values of search result *Out*, found element number *InOutPos*, and number found *Num* are as follows:

Element with same value as *Key*	Out	InOutPos	Num
Exists.	TRUE	Lowest element number that contains the same value as *Key*	Number of elements with same value as *Key*
Does not exist.	FALSE	Does not change.	0

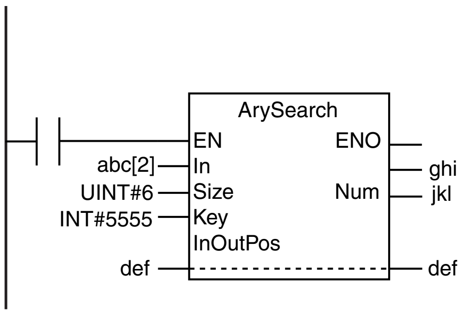
The relationship between values with data types that are not integers or real numbers are determined as given in the following table.

Data type	Relationship
TIME	The numerically larger value is considered to be larger.
DATE, TOD, or DT	Later dates or times of day are considered to be larger.

The following example is for when *Size* is UINT#6 and *Key* is INT#5555.

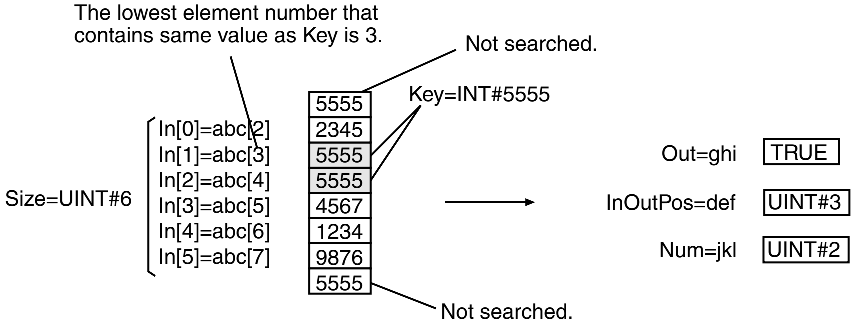
The input parameter that is passed to *In[]* is *abc[2]*, so the search starts from *abc[2]*.

LD



ST

```
ghi:=ArySearch(abc[2], UINT#6, INT#5555, def, jkl);
```



Additional Information

When you compare TIME, DT, or TOD data, adjust the data so that the precision of the values is the same. Use the following instructions to adjust the precision of the values: TruncTime (page 2-657), TruncDt (page 2-661), and TruncTod (page 2-665).

Precautions for Correct Use

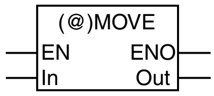
- Always use a one-dimensional array for *In[]*.
- Make sure that *Key* has the same data type as the elements of *In[]*.
- If the value of *Size* is 0, the values of *Out* and *Num* are 0. The value of *InOutPos* does not change.
- Always use a variable for the input parameter to pass to *Key*. A building error will occur if a constant is passed.
- If *Key* is an enumeration, you cannot directly pass an enumerator to it. A building error will occur if an enumerator is passed to it directly.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out*, *Num*, and *InOutPos* will not change.
  - *Size* exceeds the array area of *In[]*.
  - *In[]* is STRING data and it does not end in a NULL character.
  - *In[]* is not a one-dimensional array.

Data Movement Instructions

Instruction	Name	Page
MOVE	Move	2-354
MoveBit	Move Bit	2-357
MoveDigit	Move Digit	2-359
TransBits	Move Bits	2-361
MemCopy	Memory Copy	2-363
SetBlock	Block Set	2-365
Exchange	Data Exchange	2-367
AryExchange	Array Data Exchange	2-369
AryMove	Array Move	2-371
Clear	Initialize	2-373
Copy**ToNum (Bit String to Signed Integer)	Bit Pattern Copy (Bit String to Signed Integer) Group	2-375
Copy**To*** (Bit String to Real Number)	Bit Pattern Copy (Bit String to Real Number) Group	2-377
CopyNumTo** (Signed Integer to Bit String)	Bit Pattern Copy (Signed Integer to Bit String) Group	2-379
CopyNumTo** (Signed Integer to Real Number)	Bit Pattern Copy (Signed Integer to Real Number) Group	2-381
Copy**To*** (Real Number to Bit String)	Bit Pattern Copy (Real Number to Bit String) Group	2-383
Copy**ToNum (Real Number to Signed Integer)	Bit Pattern Copy (Real Number to Signed Integer) Group	2-385

MOVE

The MOVE instruction moves the value of a constant or variable to another variable.

Instruction	Name	FB/FUN	Graphic expression	ST expression
MOVE	Move	FUN		Out:=In;

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Move source	Input	Move source	Depends on data type.	---	*
Out	Move destination	Output	Move destination	Depends on data type.	---	*

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	An enumeration, array, array element, structure, or structure member can also be specified.																			
Out	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	Must be the same data type as *In* if *In* is an enumeration, array element, structure, or structure member. Must be an array with the same data type, size, and subscripts if *In* is an array.																			

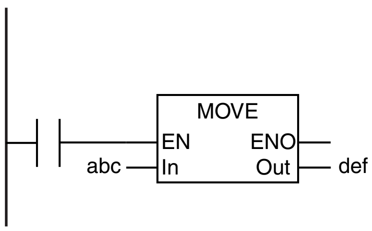
Function

The MOVE instruction moves the value in move source *In* to move destination *Out*. The input parameter that is passed to *In* can be a variable or constant. You can specify an enumeration, array, array element, structure, or structure member for *In*.

The following figure shows a programming example. The content of variable *abc* is moved to variable *def.*

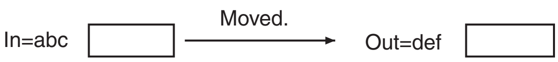
LD

ST



def:=abc;

The MOVE instruction moves the value of *In* to *Out*.

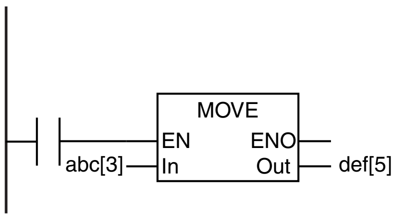


Additional Information

- When moving an array, you can move either one element or all of the elements in the array. To move only one element, add the subscript to the array variable name. To move the entire array, do not add the subscript to the array variable name.

Moving One Array Element

LD

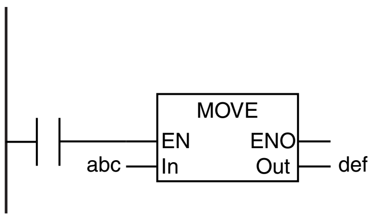


ST

```
def[5]:=abc[3];
```

Moving All Array Elements

LD



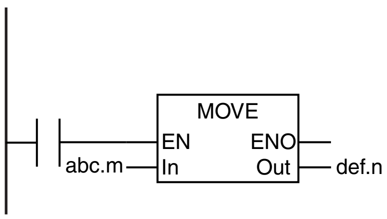
ST

```
def:=abc;
```

- When moving a structure, you can move either one member or all of the members in the structure. To move only one member, specify the member. To move the entire structure, give only the structure name.

Moving One Member of a Structure

LD

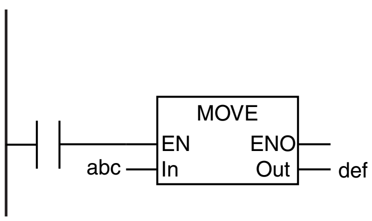


ST

```
def.n:=abc.m;
```

Moving the Entire Structure

LD



ST

```
def:=abc;
```

- You can use the MemCopy instruction to move an entire array faster than with the MOVE instruction.

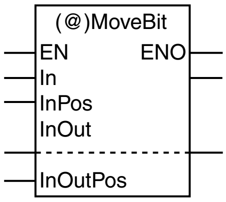
Precautions for Correct Use

- The data types of *In* and *Out* can be different as long as they are both in one of the following groups. The valid range of *Out* must include the valid range of *In*.

  - BYTE, WORD, DWORD, and LWORD
  - USINT, UINT, UDINT, ULINT, SINT, INT, DINT, LINT, REAL, and LREAL
- If *In* is an enumeration, array element, structure, or structure member, then *Out* must have the same data type as *In*.
- If *In* is an array, an array of the same data type, size, and subscripts must be used for *Out*.

MoveBit

The MoveBit instruction moves one bit in a bit string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
MoveBit	Move Bit	FUN		MoveBit(In, InPos, InOut, InOutPos);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Move source	Input	Move source	Depends on data type.	---	*
InPos	Move source bit	Input	Position of bit in *In* to move	0 to No. of bits in *In* – 1	---	0
InOutPos	Move destination bit	Input	Position of bit in *Out* to receive the bit	0 to No. of bits in *InOut* – 1	---	0
InOut	Move destination	In-out	Move destination	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

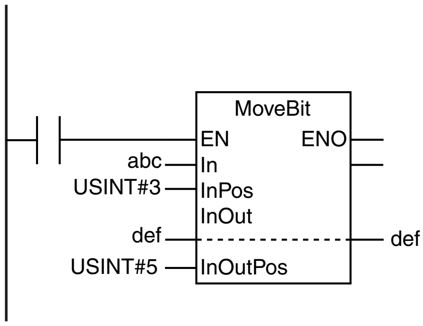
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
InPos						OK														
InOutPos						OK														
InOut		OK	OK	OK	OK															
Out	OK																			

Function

The MoveBit instruction moves one bit from the bit position *InPos* in move source *In* to the bit position *InOutPos* in move destination *InOut*.

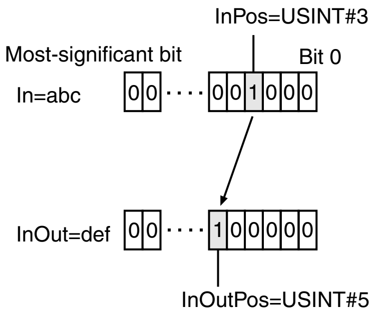
The following example is for when *InPos* is USINT#3 and *InOutPos* is USINT#5.

LD



ST

```
MoveBit(abc, USINT#3, def, USINT#5);
```

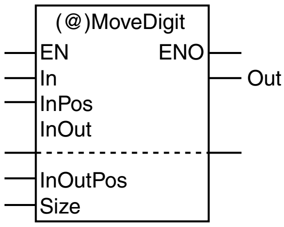


Precautions for Correct Use

- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *InOut* will not change.
  - The value of *InPos* is outside of the valid range.
  - The value of *InOutPos* is outside of the valid range.

MoveDigit

The MoveDigit instruction moves digits (4 bits per digit) in a bit string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
MoveDigit	Move Digit	FUN		MoveDigit(In, InPos, InOut, InOutPos, Size);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Move source	Input	Move source	Depends on data type.	---	*1
InPos	Move source digit		Position of digit in *In* to move	*2		0
InOutPos	Move destination digit		Position of digit in *Out* to receive the digit	*3		
Size	Number of digits		Number of digits to move	*4		1
InOut	Move destination	In-out	Move destination	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

*1 If you omit the input parameter, the default value is not applied. A building error will occur.

*2 0 to No. of bits in *In*/4 – 1

*3 0 to No. of bits in *InOut*/4 – 1

*4 0 to No. of bits in *In*/4

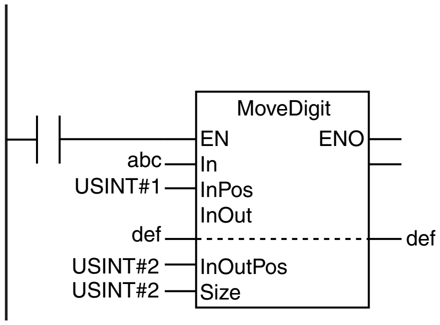
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
InPos						OK														
InOutPos						OK														
Size						OK														
InOut		OK	OK	OK	OK															
Out	OK																			

Function

The MoveDigit instruction moves *Size* digits from the *InPos* digit in move source *In* to the *InOutPos* digit in move destination *InOut*. One digit is four bits.

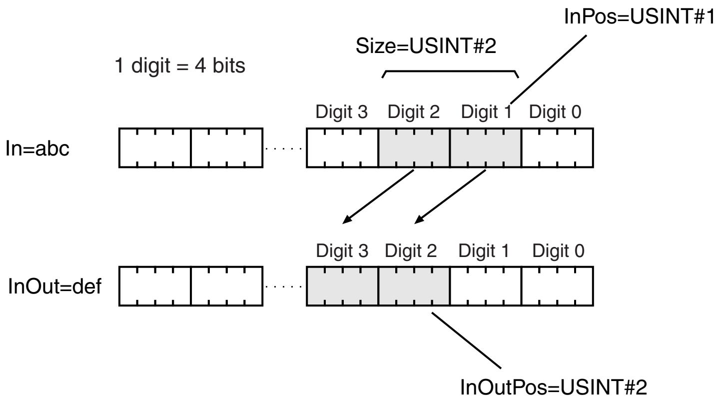
The following example is for when *InPos* is USINT#1, *InOutPos* is USINT#2, and *Size* is USINT#2.

LD



ST

```
MoveDigit(abc, USINT#1, def, USINT#2, USINT#2);
```

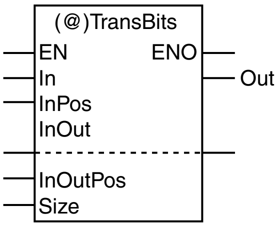


Precautions for Correct Use

- If the position of the digit at the destination exceeds the most-significant digit of *InOut*, the remaining digits are stored the least-significant digits of *InOut*.
- If the position of the digit at the source exceeds the most-significant digit of *In*, the remaining digits are moved to the least-significant digits of *In*.
- If the value of *Size* is 0, the value of *Out* will be TRUE and *InOut* will not change.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *InOut* will not change.
  - The value of *InPos* is outside of the valid range.
  - The value of *InOutPos* is outside of the valid range.
  - The value of *Size* is outside of the valid range.

TransBits

The TransBits instruction moves one or more bits in a bit string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TransBits	Move Bits	FUN		TransBits(In, InPos, InOut, InOutPos, Size);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Move source	Input	Move source	Depends on data type.	---	*1
InPos	Move source bit		Position of bit in *In* to move	*2		0
InOutPos	Move destination bit		Position of bit in *Out* to receive the bit	*3		
Size	Number of bits		Number of bits to move	*4		1
InOut	Move destination	In-out	Move destination	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

*1 If you omit an input parameter, the default value is not applied. A building error will occur.

*2 0 to No. of bits in *In* – 1

*3 0 to No. of bits in *InOut* – 1

*4 0 to No. of bits in *In*

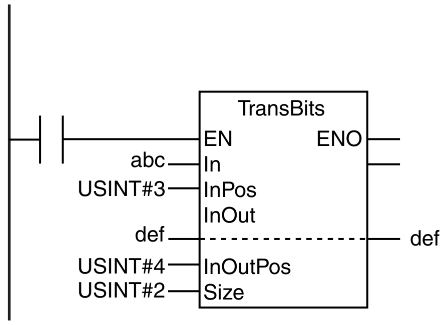
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
InPos						OK														
InOutPos						OK														
Size						OK														
InOut		OK	OK	OK	OK															
Out	OK																			

Function

The TransBis instruction moves *Size* bits from the *InPos* bit in move source *In* to the *InOutPos* bit in move destination *InOut*.

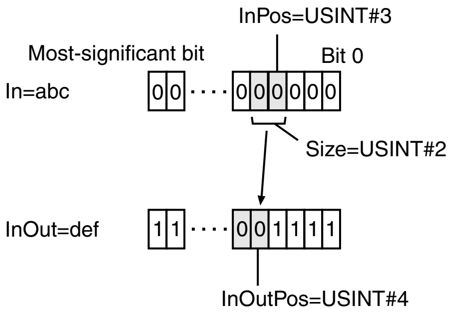
The following example is for when *InPos* is USINT#3, *InOutPos* is USINT#4, and *Size* is USINT#2.

LD



ST

```
TransBits(abc, USINT#3, def, USINT#4, USINT#2);
```



Additional Information

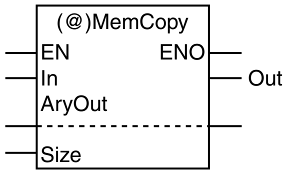
The bits in the move source and move destination can overlap.

Precautions for Correct Use

- Set the instruction so that the positions of the bits at the source and destination do not exceed the most-significant bit in *In* or *InOut*. An error will occur and the instruction will not operate.
- Nothing is moved if the value of *Size* is 0.
- The bits in *InOut* that are not involved in the move operation do not change.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *InOut* will not change.
  - The value of *InPos* is outside of the valid range.
  - The value of *InOutPos* is outside of the valid range.
  - The value of *Size* is outside of the valid range.
  - The value of *InPos* or *Size* exceeds the number of bits in *In*.
  - The value of *InOutPos* or *Size* exceeds the number of bits in *InOut*.

MemCopy

The MemCopy instruction moves one or more array elements. The move source and move destination must have the same data type.

Instruction	Name	FB/FUN	Graphic expression	ST expression
MemCopy	Memory Copy	FUN		MemCopy(In, AryOut, Size);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Move source array	Input	Move source array	Depends on data type.	---	*
Size	Number of elements		Number of array elements to move			1
AryOut[] (array)	Move destination array	In-out	Move destination array	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

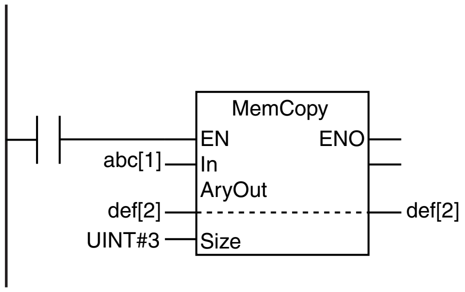
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	Arrays of enumerations or structures can also be specified.																			
Size							OK													
AryOut[] (array)	Must be an array with the same data type as *In[]*.																			
Out	OK																			

Function

The MemCopy instruction moves *Size* elements of move source array *In[]* starting from *In[0]* to move destination array *AryOut[]* starting from *AryOut[0]*.

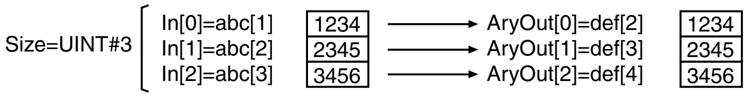
The following example is for when *Size* is UINT#3.

LD



ST

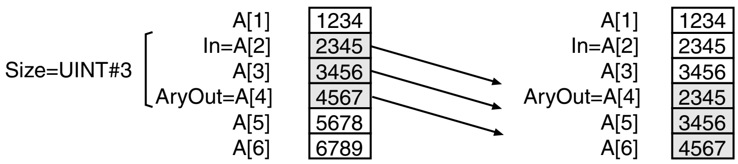
```
MemCopy(abc[1], def[2], UINT#3);
```



Additional Information

- You can specify different positions in the same array for *In[]* and *AryOut[]*. The source and destination data can overlap.

The following example is for when *In* is *A[2]*, *AryOut* is *A[4],* and *Size* is UINT#3.



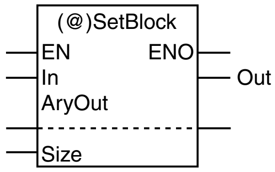
- Use the AryMove instruction (page 2-371) if the source and destination have different data types.
- If the data types of *In[]* and *AryOut[]* are the same, this instruction is faster than the AryMove instruction.
- Use the MOVE instruction (page 2-354) to move variables that are not arrays.

Precautions for Correct Use

- Use the same data type for *In[]* and *AryOut[].* If they are different, a building error will occur.
- If *In[]* and *AryOut[]* are STRING arrays, their sizes must be the same.
- If the value of *Size* is 0, the value of *Out* will be TRUE and *AryOut[]* will not change.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *AryOut[]* will not change.
  - *Size* exceeds the array area of *In[]*.
  - *Size* exceeds the array area of *AryOut[]*.

SetBlock

The SetBlock instruction moves the value of a variable or constant to one or more array elements.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SetBlock	Block Set	FUN		SetBlock(In, AryOut, Size);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Move source	Input	Move source	Depends on data type.	---	*
Size	Number of elements		Number of array elements to move			1
AryOut[] (array)	Move destination array	In-out	Move destination array	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

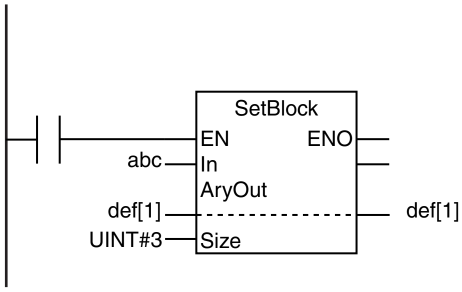
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	An enumeration, structure, or structure member can also be specified.																			
Size							OK													
AryOut[] (array)	Must be an array with elements that have the same data type as *In*.																			
Out	OK																			

Function

The SetBlock instruction moves the value of move source *In* to *Size* locations in move destination array *AryOut[]* starting from *AryOut[0]*.

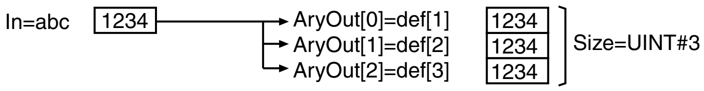
The following example is for when *Size* is UINT#3.

LD



ST

```
SetBlock(abc, def[1], UINT#3);
```

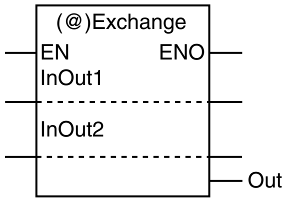


Precautions for Correct Use

- Use the same data type for *In* and *AryOut[]*. If they are different, a building error will occur.
- If *In* and *AryOut[]* are STRING data, their sizes must be the same.
- If the value of *Size* is 0, the value of *Out* will be TRUE and *AryOut[]* will not change.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following case. *ENO* will be FALSE, and *AryOut[]* will not change.
    - The value of *Size* exceeds the array area of *AryOut[]*.

Exchange

The Exchange instruction exchanges the values of two variables.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Exchange	Data Exchange	FUN		Exchange(InOut1, InOut2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
InOut1 and InOut2	Data to exchange	In-out	Data to exchange	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

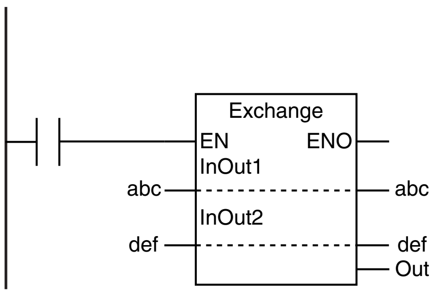
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
InOut1	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	An enumeration, structure, or structure member can also be specified.																			
InOut2	Must be same data type as *InOut1*.																			
Out	OK																			

Function

The Exchange instruction exchanges the values of data to exchange *InOut1* and *InOut2.* You can specify enumerations, structures, or structure members for *InOut1* and *InOut2*.

The following figure shows a programming example. The values in variables *abc* and *def* are exchanged.

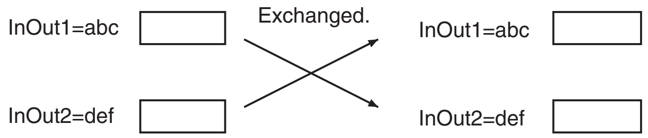
LD



ST

```
Exchange(abc, def);
```

The Exchange instruction exchanges the values of *InOut1* and *InOut2*.

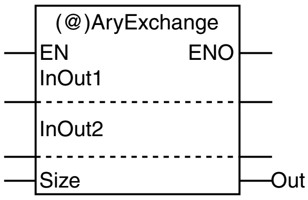


Precautions for Correct Use

- The data types of *InOut1* and *InOut2* must be the same. If they are different, a building error will occur.
- If the regions specified by *InOut1* and *InOut2* overlap each other, the execution result of the instruction will be undefined.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *InOut1* and *InOut2* will not change.
  - Both *InOut1* and *InOut2* are STRING data and the length of the text string in one of them does not fit into the other.

AryExchange

The AryExchange instruction exchanges the elements of two arrays.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryExchange	Array Data Exchange	FUN		AryExchange(InOut1, InOut2, Size);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Size	Number of elements	Input	Number of elements to exchange	Depends on data type.	---	1
InOut1[] and InOut2[] (arrays)	Arrays to exchange	In-out	Arrays to exchange	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

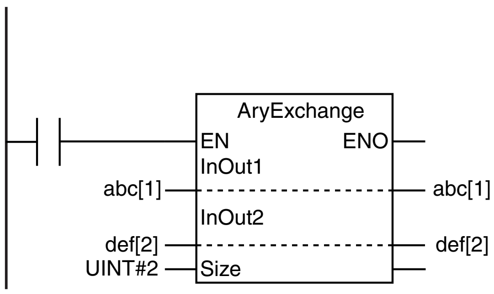
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Size							OK													
InOut1[] (array)	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	Arrays of enumerations or structures can also be specified.																			
InOut2[] (array)	Must be an array with the same data type as *InOut1[]*.																			
Out	OK																			

Function

The AryExchange instruction exchanges *Size* elements from *InOut1[0]* of array to exchange *InOut1[]* with *Size* elements from *InOut2[0]* of array to exchange *InOut2[]*.

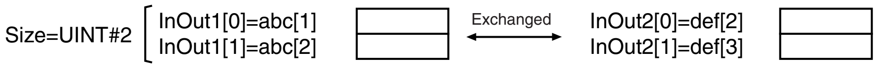
The following example is for when *Size* is UINT#2.

LD



ST

```
AryExchange(abc[1], def[2], UINT#2);
```



Additional Information

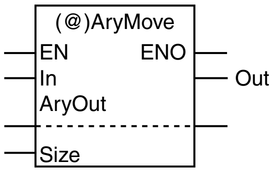
- Use the MOVE instruction (page 2-354) to assign constants to variables.
- Use the MemCopy instruction (page 2-363) to copy the values of variables to other variables.

Precautions for Correct Use

- Use the same data type for the elements of *InOut1[]* and *InOut2[]*. If they are different, a building error will occur.
- If the value of *Size* is 0, the value of *Out* will be TRUE and *InOut1[]* and *InOut2[]* will not change.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *InOut1[]* and *InOut2[]* will not change.
  - The value of *Size* exceeds the array range of *InOut1[]* or *InOut2[]*.
  - *InOut1[]* and *InOut2[]* are STRING arrays and there is an element with a text string that exceeds the size of the element in the other array.
  - *InOut1[]* and *InOut2[]* are STRING arrays and there is an element that does not end in a NULL character.

AryMove

The AryMove instruction moves one or more array elements. The data types of the move source and move destination can be different.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryMove	Array Move	FUN		AryMove(In, AryOut, Size);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Move source array	Input	Array to move	Depends on data type.	---	*
Size	Number of elements		Number of elements to move			1
AryOut[] (array)	Move result array	In-out	Move result array	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

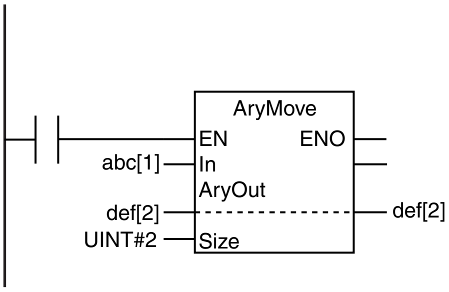
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	Arrays of enumerations or structures can also be specified.																			
Size							OK													
AryOut[] (array)	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	Arrays of enumerations or structures can also be specified.																			
Out	OK																			

Function

The *AryMove* instruction moves *Size* elements of move source array *In[]* starting from *In[0]* to move result array *AryOut[]* starting from *AryOut[0]*. The data types of *In[]* and *AryOut[]* can be different.

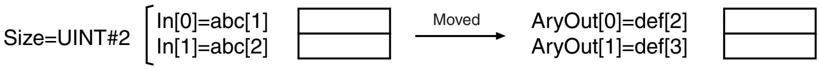
The following example is for when *Size* is UINT#2.

LD



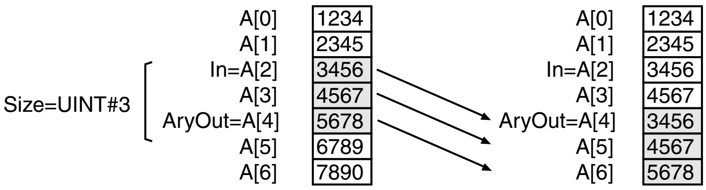
ST

```
AryMove(abc[1], def[2], UINT#2);
```



Additional Information

- If the data types of *In[]* and *AryOut[]* are the same, the MemCopy instruction is faster.
- You can specify the same array for *In[]* and *AryOut[]*. Also, the move source and destination data can overlap. The following example is for when *In[0]* is *A[2]*, *AryOut[0]* is *A[4],* and *Size* is UINT#3.



Precautions for Correct Use

- The data types of *In[]* and *AryOut[]* can be different as long as they are both in one of the following groups. The valid range of *AryOut[]* must include the valid range of *In[]*.
  - BYTE, WORD, DWORD, and LWORD
  - USINT, UINT, UDINT, ULINT, SINT, INT, DINT, LINT, REAL, and LREAL
- If *In[]* is an array of structures, use the same data types for *In[]* and *AryOut[]*.
- If the value of *Size* is 0, the value of *Out* will be TRUE and *AryOut[]* will not change.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following case. *ENO* will be FALSE, and *AryOut[]* will not change.
  - The value of *Size* exceeds the size of *In[]* or Ary*Out[]*.
  - *In[]* or *AryOut[]* is a STRING array and the length of a text string in an element to move exceeds the size of the element in *AryOut[]*.

Clear

The Clear instruction initializes a variable.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Clear	Initialize	FUN	(@)Clear EN ENO InOut Out	Clear(InOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
InOut	Data to initialize	In-out	Data to initialize	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
InOut	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	An enumeration, array, array element, structure, or structure member can also be specified.																			
Out	OK																			

Function

The Clear instruction initializes the value of data to initialize *InOut*.

If an initial value attribute is set for a variable, the specified initial value is used. If an initial value attribute is not set, the default initial value for the data type of *InOut* is used. If *InOut* is an external variable, the default initial value of the data type of *InOut* is used regardless of the initial value attribute of the global variable for the external variable.

The default values for the data types are given below.

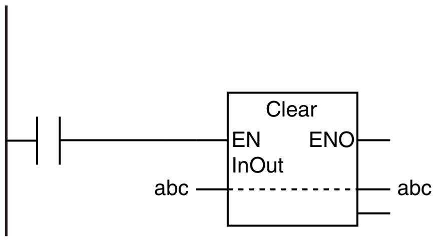
Data type	Default initial value
BOOL	FALSE
BYTE, WORD, DWORD, or LWORD	16#0
USINT, UINT, UDINT, ULINT, SINT, INT, DINT, LINT, REAL, or LREAL	0
TIME	T#0ms
DATE	D#1970-1-1
TOD	TOD#0:0:0
DT	DT#1970-1-1-0:0:0
STRING	''

If *InOut* is an array, array element, structure, or structure member, the following processing is performed.

InOut	Processing
Array	All elements in the array are initialized.
Array element	Only the specified element is initialized.
Structure	All members in the structure are initialized.
Structure member	Only the specified member is initialized.

The following figure shows a programming example. The value of variable *abc* is initialized.

LD



ST

```
Clear(abc);
```

The Clear instruction initializes the value of *InOut*.
The data type of *abc* is INT, so the value of *abc* will be INT#0.

InOut=abc $\boxed{\text{INT\#100}} \xrightarrow{\text{Initialized.}}$ InOut=abc $\boxed{\text{INT\#0}}$

Additional Information

- If *InOut* is an array that is used as a stack, execute this instruction and also set the variable that manages the number of items stored in the stack to 0.
- If you initialize a cam data variable with this instruction, it will not contain the data that was saved with the MC_SaveCamTable instruction. It will contain all zeros.

Precautions for Correct Use

- Return value *Out* is not used when the instruction is used in ST.
- To initialize an enumerated variable, use the Initial Value attribute. If the Initial Value attribute is not set, the value of the enumerated variable will be 0.
- Do not perform processing that meets all of the following conditions. The operation is not reliable.
  - Pass one element of a BOOL array as an in-out variable to a function or function block.
  - Execute the Clear instruction in the function or function block.
  - Use the in-out variable that received the element of the above BOOL array as the parameter to pass to the Clear instruction.

Copy**ToNum (Bit String to Signed Integer)

The Copy**ToNum instruction copies the content of a bit string directly to a signed integer.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Copy**ToNum	Bit Pattern Copy (Bit String to Signed Integer) Group	FUN	(@)Copy**ToNum EN ENO In Out "**" must be a bit string data type.	Out:=Copy**ToNum(In); "**" must be a bit string data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Copy source	Input	Copy source	Depends on data type.	---	0
Out	Copy destination	Output	Copy destination	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Out	Must be a signed integer data type that is the same size as the data type of *In*.																			

Function

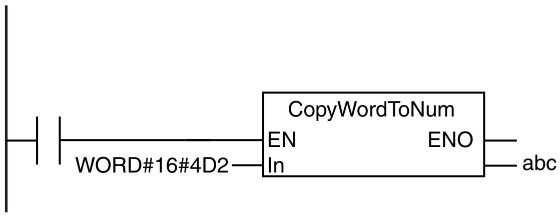
The Copy**ToNum instruction copies the content of copy source *In* directly to copy destination *Out*.

There are four instructions depending on the data types of *In* and *Out*.

In	*Out*	Instruction
BYTE	SINT	CopyByteToNum
WORD	INT	CopyWordToNum
DWORD	DINT	CopyDwordToNum
LWORD	LINT	CopyLwordToNum

The following example for the CopyWordToNum instruction is for when *In* is WORD#16#4D2.

LD



ST

```
abc:=CopyWordToNum(WORD#16#4D2);
```

In			Out = abc	
In	WORD#16#4D2 (WORD#2#00000100_11010010)	→	Out = abc	INT#1234 (2#00000100_11010010)

Copy**To*** (Bit String to Real Number)

The Copy**To*** instruction copies the content of a bit string directly to a real number.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Copy**To***	Bit Pattern Copy (Bit String to Real Number) Group	FUN	(@)CopyDwordToReal EN ENO In Out (@)CopyLwordToLreal EN ENO In Out	Out:=CopyDwordToReal(In); or Out:=CopyLwordToLreal(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Copy source	Input	Copy source	Depends on data type.	---	0
Out	Copy destination	Output	Copy destination	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In				OK	OK															
Out	Must be REAL if the data type of *In* is DWORD and LREAL if the data type of *In* is LWORD.																			

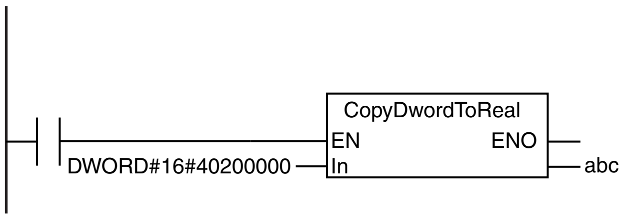
Function

The Copy**To*** instruction copies the content of copy source *In* directly to copy destination *Out*. There are two instructions depending on the data types of *In* and *Out*.

In	*Out*	Instruction
DWORD	REAL	CopyDwordToReal
LWORD	LREAL	CopyLwordToLreal

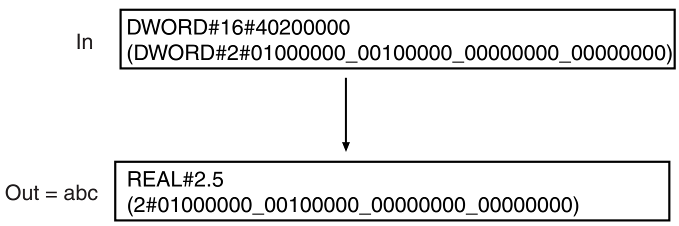
The following example for the CopyDwordToReal instruction is for when *In* is DWORD#16#40200000.

LD



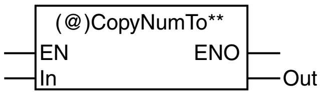
ST

```
abc:=CopyDwordToReal(DWORD#16#40200000);
```



CopyNumTo** (Signed Integer to Bit String)

The CopyNumTo** instruction copies the content of a signed integer directly to a bit string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
CopyNumTo**	Bit Pattern Copy (Signed Integer to Bit String) Group	FUN	 "**" must be a bit string data type.	Out:=CopyNumTo**(In); "**" must be a bit string data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Copy source	Input	Copy source	Depends on data type.	---	0
Out	Copy destination	Output	Copy destination	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In										OK	OK	OK	OK							
Out	Must be a bit string data type that is the same size as the data type of *In*.																			

Function

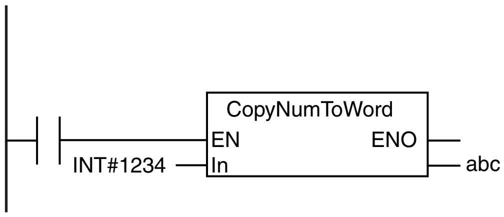
The CopyNumTo** instruction copies the content of copy source *In* directly to copy destination *Out*.

There are four instructions depending on the data types of *In* and *Out*.

In	*Out*	Instruction
SINT	BYTE	CopyNumToByte
INT	WORD	CopyNumToWord
DINT	DWORD	CopyNumToDword
LINT	LWORD	CopyNumToLword

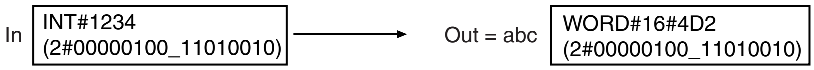
The following example for the CopyNumToWord instruction is for when *In* is INT#1234.

LD



ST

```
abc:=CopyNumToWord(INT#1234);
```



CopyNumTo** (Signed Integer to Real Number)

The CopyNumTo** instruction copies the content of a signed integer directly to a real number.

Instruction	Name	FB/FUN	Graphic expression	ST expression
CopyNumTo**	Bit Pattern Copy (Signed Integer to Real Number) Group	FUN	(@)CopyNumToReal EN ENO In Out (@)CopyNumToLreal EN ENO In Out	Out:=CopyNumToReal(In); or Out:=CopyNumToLreal(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Copy source	Input	Copy source	Depends on data type.	---	0
Out	Copy destination	Output	Copy destination	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In												OK	OK							
Out	Must be REAL if the data type of *In* is DINT and LREAL if the data type of *In* is LINT.																			

Function

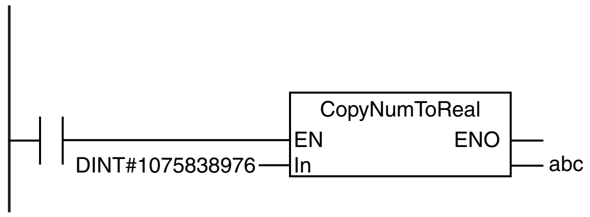
The CopyNumTo** instruction copies the content of copy source *In* directly to copy destination *Out*.

There are two instructions depending on the data types of *In* and *Out*.

In	*Out*	Instruction
DINT	REAL	CopyNumToReal
LINT	LREAL	CopyNumToLreal

The following example for the CopyNumToReal instruction is for when *In* is DINT#1075838976.

LD



ST

```
abc:=CopyNumToReal(DINT#1075838976);
```

In: DINT#1075838976 (2#01000000_00100000_00000000_00000000) → Out = abc: REAL#2.5 (2#01000000_00100000_00000000_00000000)

Copy**To*** (Real Number to Bit String)

The Copy**To*** instruction copies the content of a real number directly to a bit string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Copy**To***	Bit Pattern Copy (Real Number to Bit String) Group	FUN	(@)CopyRealToDword EN ENO In Out (@)CopyLrealToLword EN ENO In Out	Out:=CopyRealToDword(In); or Out:=CopyLrealToLword(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Copy source	Input	Copy source	Depends on data type.	---	0.0
Out	Copy destination	Output	Copy destination	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK	OK					
Out	Must be DWORD if the data type of *In* is REAL and LWORD if the data type of *In* is LREAL.																			

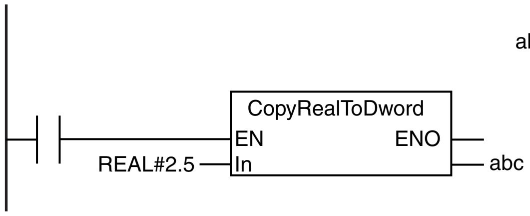
Function

The Copy**To*** instruction copies the content of copy source *In* directly to copy destination *Out*.
There are two instructions depending on the data types of *In* and *Out*.

In	*Out*	Instruction
REAL	DWORD	CopyRealToDword
LREAL	LWORD	CopyLrealToLword

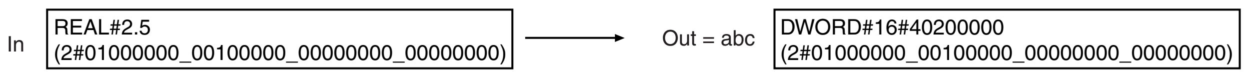
The following example for the CopyRealToDword instruction is for when *In* is REAL#2.5.

LD



ST

```
abc:=CopyRealToDword(REAL#2.5);
```



Copy**ToNum (Real Number to Signed Integer)

The Copy**ToNum instruction copies the content of a real number directly to a signed integer.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Copy**ToNum	Bit Pattern Copy (Real Number to Signed Integer) Group	FUN	(@)CopyRealToNum: EN, In → ENO, Out (@)CopyLrealToNum: EN, In → ENO, Out	Out:=CopyRealToNum(In); or Out:=CopyLrealToNum(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Copy source	Input	Copy source	Depends on data type.	---	0.0
Out	Copy destination	Output	Copy destination	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK	OK					
Out	Must be DINT if the data type of *In* is REAL and LINT if the data type of *In* is LREAL.																			

Function

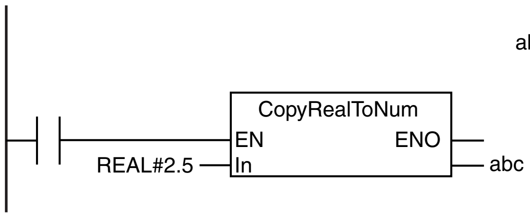
The Copy**ToNum instruction copies the content of copy source *In* directly to copy destination *Out*.

There are two instructions depending on the data types of *In* and *Out*.

In	*Out*	Instruction
REAL	DINT	CopyRealToNum
LREAL	LINT	CopyLrealToNum

The following example for the CopyRealToNum instruction is for when *In* is REAL#2.5.

LD



ST

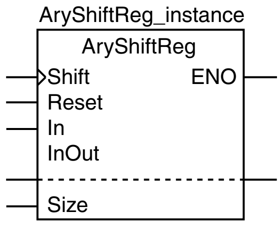
```
abc:=CopyRealToNum(REAL#2.5);
```

In: REAL#2.5 (2#01000000_00100000_00000000_00000000) → Out = abc: DINT#1075838976 (2#01000000_00100000_00000000_00000000)

Shift Instructions

AryShiftReg

The AryShiftReg instruction shifts a shift register one bit to the left and inserts the input value to the least-significant bit. The shift register consists of array elements.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryShiftReg	Shift Register	FB		AryShiftReg_instance(Shift, Reset, In, InOut, Size);

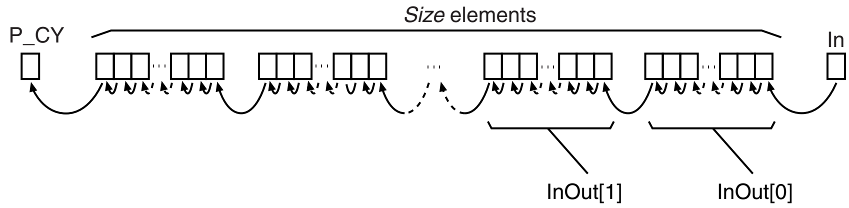
Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Shift	Shift		Shifted when signal changes to TRUE.			
Reset	Reset		TRUE: Register is reset.			FALSE
In	Input value	Input	Value to insert to least-significant bit of *InOut[]*.	Depends on data type.	---	
Size	Number of elements in array of bit strings		Number of elements to use as a shift register in *InOut[]*.			1
InOut[] (array)	Array of bit strings	In-out	Array of bit strings	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Shift	OK																			
Reset	OK																			
In	OK																			
Size							OK													
InOut[] (array)	OK	OK	OK	OK	OK															

Function

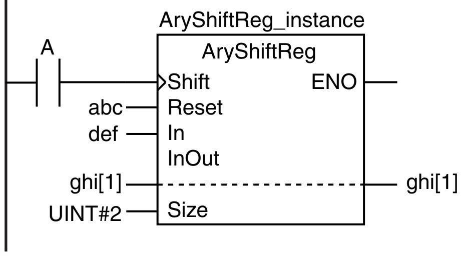
The AryShiftReg instruction shifts *Size* array elements in the array of bit strings *InOut[]* to the left (i.e., toward most-significant bit) when *Shift* changes to TRUE. The shift operation starts from *InOut[0]*. Input value *In* is inserted to the least-significant bit. The most-significant bit of the array of bit strings is output to the Carry (CY) Flag (P_CY).



When Reset is TRUE, CY and all of bits in *Size* elements starting from *InOut[0]* are set to FALSE.

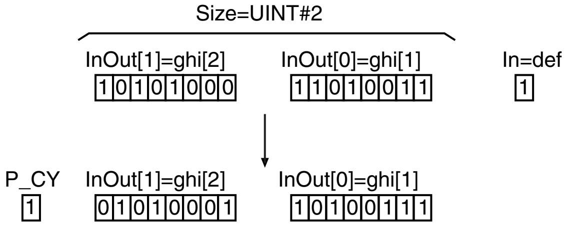
The following example is for when *InOut[]* is a BYTE array and *Size* is UINT#2.

LD



ST

AryShiftReg_instance(A, abc, def, ghi[1], UINT#2);



Related System-defined Variables

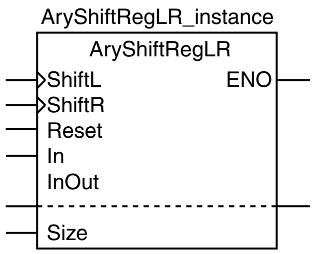
Name	Meaning	Data type	Description
P_CY	Carry (CY) Flag	BOOL	Value stored in Carry Flag

Precautions for Correct Use

- While *Reset* is TRUE, the register is not shifted even if *Shift* changes to TRUE.
- *ENO* will change to TRUE when *Shift* changes to TRUE and the shift operation is performed normally, or when *Reset* is TRUE and the reset operation is performed normally.
- The *InOut[]* does not change if the value of *Size* is 0.
- An error occurs in the following case. *ENO* will be FALSE, and *InOut[]* will not change.
  - The value of *Size* exceeds the array area of *InOut[]*.

AryShiftRegLR

The AryShiftRegLR instruction shifts a bit string one bit to the left or right and inserts the input value to the least-significant or most-significant bit. The bit string consists of array elements.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryShiftRegLR	Reversible Shift Register	FB		AryShiftRegLR_instance (ShiftL, ShiftR,Reset, In, InOut, Size);

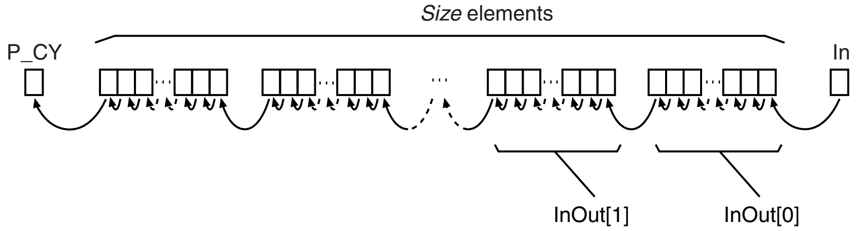
Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
ShiftL	Left shift	Input	Shifted left when signal changes to TRUE.	Depends on data type.	---	FALSE
ShiftR	Right shift		Shifted right when signal changes to TRUE.			
Reset	Reset		TRUE: Register is reset.			
In	Input value		Value to insert to least-significant or most-significant bit of *InOut[]*			
Size	Number of elements in array of bit strings		Number of elements to use as a shift register in *InOut[]*.			1
InOut[] (array)	Array of bit strings	In-out	Array of bit strings	Depends on data type.	---	---

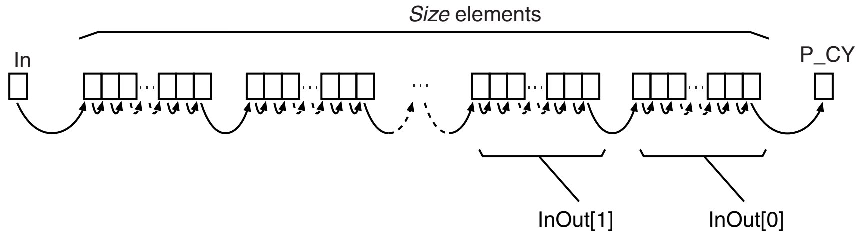
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
ShiftL	OK																			
ShiftR	OK																			
Reset	OK																			
In	OK																			
Size							OK													
InOut[] (array)	OK	OK	OK	OK	OK															

Function

The AryShiftRegLR instruction shifts *Size* array elements of array of bit strings *InOut[]* to the left when *ShiftL* changes to TRUE. The shift operation starts from *InOut[0]*. Input value *In* is inserted to the least-significant bit. The most-significant bit of the array of bit strings is output to the Carry (CY) Flag (P_CY).



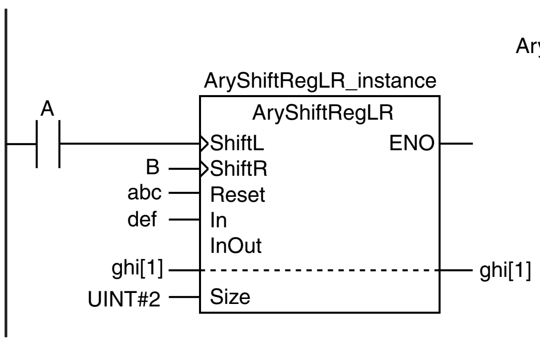
When *ShiftR* changes to TRUE, the bits are shifted one bit to the right and *In* is inserted to the most-significant bit. The least-significant bit of the array of bit strings is output to the Carry (CY) Flag (P_CY).



When Reset is TRUE, P_CY and all of the bits in *Size* elements starting from *InOut[0]* are set to FALSE.

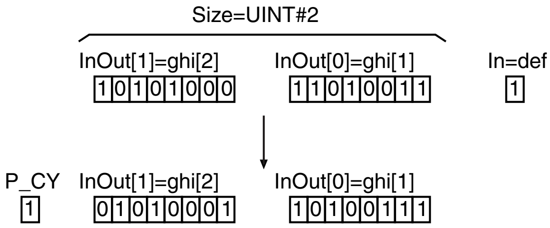
The following example is for when *InOut* is BYTE data, *Size* is UINT#2 and *ShiftL* changes to TRUE.

LD



ST

```
AryShiftRegLR_instance(A, B, abc, def, ghi[1], UINT#2);
```



Related System-defined Variables

Name	Meaning	Data type	Description
P_CY	Carry (CY) Flag	BOOL	Value stored in Carry Flag

Precautions for Correct Use

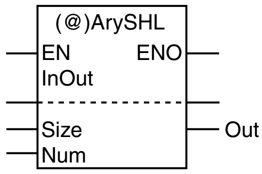
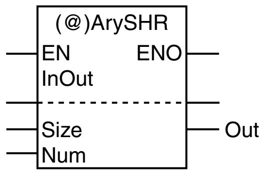
- While *Reset* is TRUE, the register is not shifted even if *ShiftL* or *ShiftR* changes to TRUE.
- The register is not shifted if both *ShiftL* and *ShiftR* change to TRUE at the same time.
- *ENO* will change to TRUE when *ShiftL* or *ShiftR* changes to TRUE and the shift operation is performed normally, or when *Reset* is TRUE and the reset operation is performed normally.
- The *InOut[]* does not change if the value of *Size* is 0.
- An error occurs in the following case. *ENO* will be FALSE, and *InOut[]* will not change.
  - The value of *Size* exceeds the array area of *InOut[]*.

AryShL and AryShR

These instructions shift array elements by one or more elements.

AryShL: Shifts the array to the left (toward the higher elements).

AryShR: Shifts the array to the right (toward the lower elements).

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryShL	Array N-element Left Shift	FUN		AryShL(InOut, Size, Num);
AryShR	Array N-element Right Shift	FUN		AryShR(InOut, Size, Num);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Size	Number of elements in shift register	Input	Number of elements in shift register	Depends on data type.	---	1
Num	Number of elements to shift		Number of elements to shift			
InOut[] (array)	Shift register array	In-out	Shift register array	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Size							OK													
Num							OK													
InOut[] (array)	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	Arrays of structures can also be specified.																			
Out	OK																			

Function

These instructions shift the upper *Size* elements in shift register array *InOut[]* by *Num* elements.
The values that are shifted out of the array are discarded.
The default initial value for the data type of *InOut[]* is stored in the empty elements.
If *InOut[]* is an array of structures, the members of the structures in all elements are initialized.
The default values for the data types are given below.

Data type of *InOut*	Default
BOOL	FALSE
BYTE, WORD, DWORD, or LWORD	16#0
USINT, UINT, UDINT, ULINT, SINT, INT, DINT, LINT, REAL, or LREAL	0
TIME	T#0ms
DATE	D#1970-1-1
TOD	TOD#0:0:0
DT	DT#1970-1-1-0:0:0
STRING	''

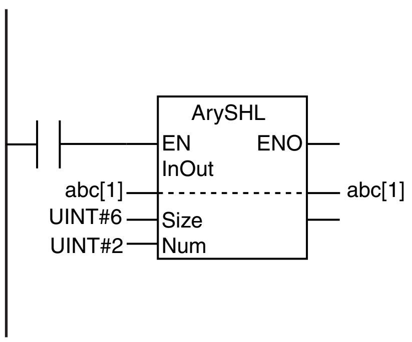
● ArySHL

The ArySHL instruction shifts the array to the left (toward the higher elements of the array).

● ArySHR

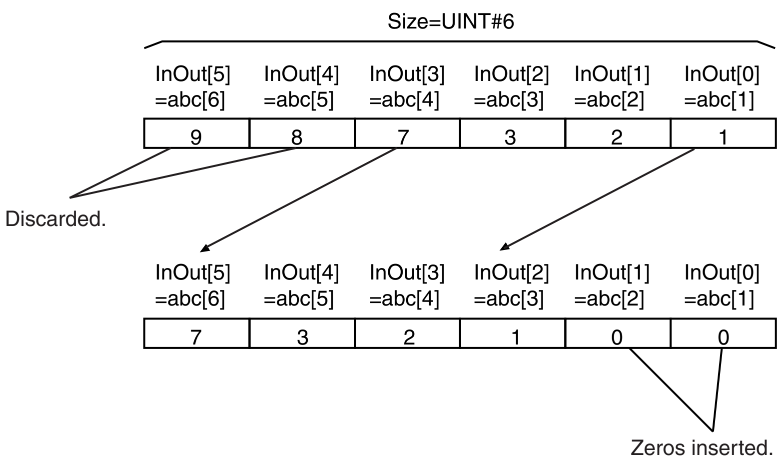
The ArySHR instruction shifts the array to the right (toward the lower elements of the array).
The following example shows the ArySHL instruction when *Size* is UINT#6 and *Num* is UINT#2.

LD



ST

```
ArySHL(abc[1], UINT#6, UINT#2);
```



Additional Information

If *InOut[]* is BOOL data, the results will be the same as shifting a bit string of *Size* bits by *Num* bits.

Precautions for Correct Use

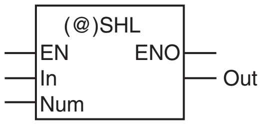
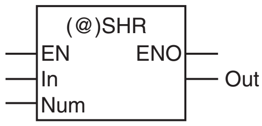
- The shift operation is not performed if the value of *Num* is 0.
- If the value of *Num* is larger than *Size*, all values from *InOut[0]* to *InOut[Size*–1] are initialized.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following case. *ENO* will be FALSE, and *InOut[]* will not change.
  - The value of *Size* exceeds the array area of *InOut[]*.

SHL and SHR

These instructions shift a bit string by one or more bits.

SHL: Shifts the bit string to the left (toward the higher bits).

SHR: Shifts the bit string to the right (toward the lower bits).

Instruction	Name	FB/FUN	Graphic expression	ST expression
SHL	N-bit Left Shift	FUN		Out:=SHL(In, Num);
SHR	N-bit Right Shift	FUN		Out:=SHR(In, Num);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to shift	Input	Data to shift	Depends on data type.	---	*1
Num*2	Number to shift		Number of bits to shift	0 to No. of bits in *In*	Bits	1
Out	Processing result	Output	Processing result	Depends on data type.	---	---

*1 If you omit the input parameter, the default value is not applied. A building error will occur.

*2 On Sysmac Studio version 1.03, you can use “N” instead of “Num” to more clearly show the correspondence between the variables and the parameter names in ST expressions. For example, you can use the following notation: *Out:=SHL*(*In:=BYTE#16#89, N:=ULINT#2*);.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Num						OK *			OK *											
Out	Must be same data type as *In*																			

* With a CPU Unit with unit version 1.02 or later and Sysmac Studio version 1.03 or higher, use a ULINT variable. With a CPU Unit with unit version 1.01 or earlier and Sysmac Studio version 1.02 or lower, use a USINT variable.

Function

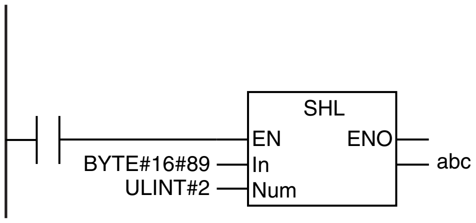
These instructions shift data to shift *In* (bit string data) by the number of bits specified in number to shift *Num*. The bits that are shifted out of the register are discarded and zeros are inserted into the other end of the register.

● SHL

The SHL instruction shifts bits from right to left (from least-significant to most-significant bits).

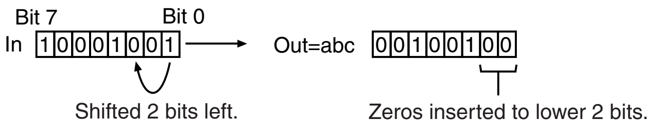
The following example is for when *In* is BYTE#16#89 and *Num* is ULINT#2.

LD



ST

abc:=SHL(BYTE#16#89, ULINT#2);

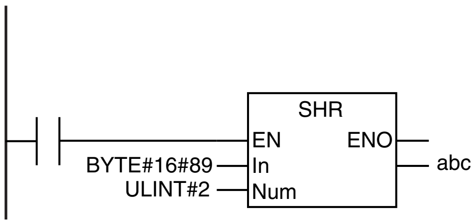


● SHR

The SHR instruction shifts bits from left to right (from most-significant to least-significant bits).

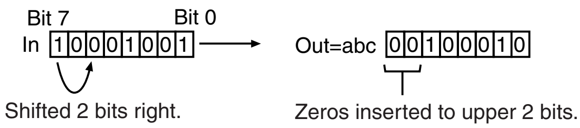
The following example is for when *In* is BYTE#16#89 and *Num* is ULINT#2.

LD



ST

abc:=SHR(BYTE#16#89, ULINT#2);



Additional Information

The ROL and ROR instructions insert the bits that are shifted out of the register into the other end of the register.

Precautions for Correct Use

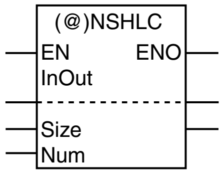
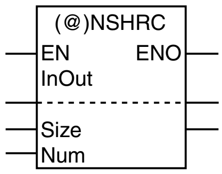
- The data types of *In* and *Out* must be the same.
- If *Num* is 0, an error will not occur and the value of *In* will be assigned directly to *Out*.
- If the value of *Num* exceeds the number of bits specified in *In*, an error will not occur and the value of *Out* will be 16#0.

NSHLC and NSHRC

These instructions shift an array of bit strings by one or more bits. The Carry (CY) Flag is included.

NSHLC: Shifts the array to the left (toward the higher elements).

NSHRC: Shifts the array to the right (toward the lower elements).

Instruction	Name	FB/FUN	Graphic expression	ST expression
NSHLC	Shift N-bits Left with Carry	FUN		NSHLC(InOut, Size, Num);
NSHRC	Shift N-bits Right with Carry	FUN		NSHRC(InOut, Size, Num);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Size	Number of bits in shift register	Input	Number of bits in shift register	Depends on data type.	Bits	1
Num	Number of bits to shift		Number of bits to shift			
InOut[] (array)	Shift register array	In-out	Bit string array to shift	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Size						OK														
Num						OK														
InOut[] (array)	OK	OK	OK	OK	OK															
Out	OK																			

Function

These instructions shift *Size* array elements in shift register array *InOut[]* by the number of bits specified in *Num*. The shift register starts at *InOut[0]*. The last bit that is shifted out of the register is output to the Carry (CY) Flag. Zeros are inserted for the bits at the other end.

● NSHLC

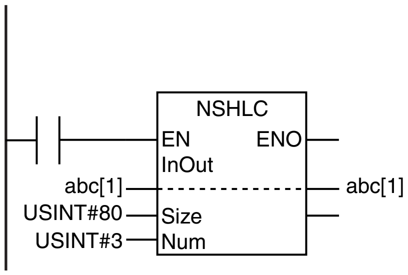
The NSHLC instruction shifts bits from the lower elements in the array to the higher elements and from the least-significant bits to the most-significant bits.

● NSHRC

The NSHRC instruction shifts bits from the higher elements in the array to the lower elements and from the most-significant bits to the least-significant bits.

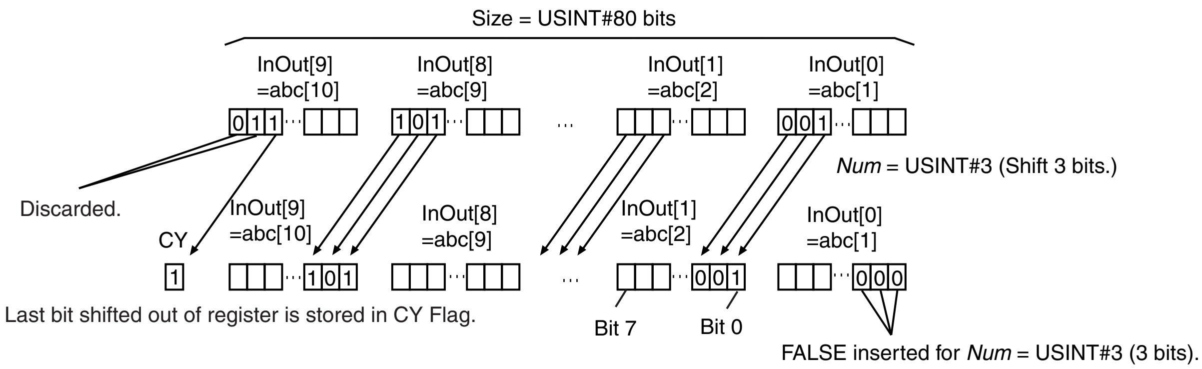
The following example shows the NSHLC instruction when *InOut[]* is a BYTE array, *Size* is USINT#80 and *Num* is USINT#3.

LD



ST

```
NSHLC(abc[1], USINT#80, USINT#3);
```



Related System-defined Variables

Name	Meaning	Data type	Description
P_CY	Carry (CY) Flag	BOOL	Value stored in Carry Flag

Precautions for Correct Use

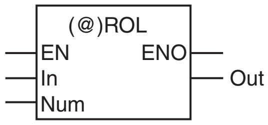
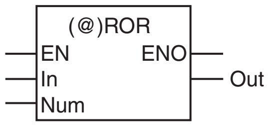
- The shift operation is not performed if the value of *Num* is 0.
- If the value of *Num* is larger than *Size*, *Size* bits from bit 0 of *InOut[0]* are changed to FALSE. The value of the Carry Flag (CY) changes to FALSE.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following case. *ENO* will be FALSE, and *InOut[]* will not change.
  - The value of *Size* exceeds the array area of *InOut[]*.

ROL and ROR

These instructions rotate a bit string by one or more bits.

ROL: Rotates the bit string to the left (toward the higher bits).

ROR: Rotates the bit string to the right (toward the lower bits).

Instruction	Name	FB/FUN	Graphic expression	ST expression
ROL	Rotate N-bits Left	FUN		Out:=ROL(In, Num);
ROR	Rotate N-bits Right	FUN		Out:=ROR(In, Num);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to rotate	Input	Data to rotate	Depends on data type.	---	*1
Num*2	Number of bits	Input	Number of bits to rotate	0 to No. of bits in *In*	Bits	1
Out	Processing result	Output	Processing result	Depends on data type.	---	---

*1 If you omit the input parameter, the default value is not applied. A building error will occur.

*2 On Sysmac Studio version 1.03, you can use “N” instead of “Num” to more clearly show the correspondence between the variables and the parameter names in ST expressions. For example, you can use the following notation: *Out:=ROL*(*In:=BYTE#16#89, N:=ULINT#2*);.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Num						OK *			OK *											
Out	Must be same data type as *In*																			

* With a CPU Unit with unit version 1.02 or later and Sysmac Studio version 1.03 or higher, use a ULINT variable. With a CPU Unit with unit version 1.01 or earlier and Sysmac Studio version 1.02 or lower, use a USINT variable.

Function

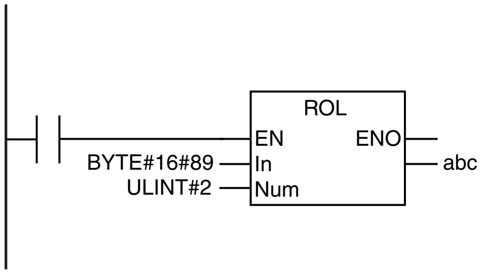
These instructions rotate data to rotate *In* (bit string data) by the number of bits specified in number of bits *Num*. Bits that are shifted out of the register are inserted into the other end of the register.

● **ROL**

The ROL instruction rotates bits from right to left (from least-significant to most-significant bits).

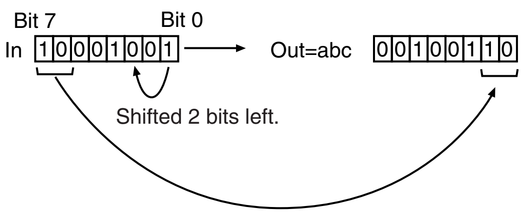
The following example is for when *In* is BYTE#16#89 and *Num* is ULINT#2.

LD



ST

```
abc:=ROL(BYTE#16#89, ULINT#2);
```



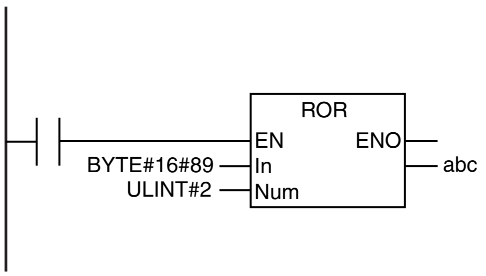
Most-significant 2 bits of *In* are inserted into least-significant 2 bits.

● **ROR**

The ROR instruction rotates bits from left to right (from most-significant to least-significant bits).

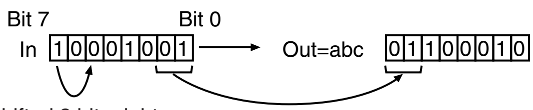
The following example is for when *In* is BYTE#16#89 and *Num* is ULINT#2.

LD



ST

```
abc:=ROR(BYTE#16#89, ULINT#2);
```



Shifted 2 bits right. Least-significant 2 bits of *In* are inserted into most-significant 2 bits.

Additional Information

The SHL and SHR instructions discard the bits that are shifted out of the register and insert zeros into the other end of the register.

Precautions for Correct Use

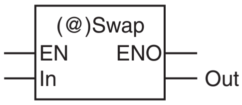
- The data types of *In* and *Out* must be the same.
- If *Num* is 0, an error will not occur and the value of *In* will be assigned directly to *Out*.
- If the value of *Num* exceeds the number of bits specified in *In*, an error will not occur and the bits will be rotated by the number of bits specified in *Num*. For example, if *In* is WORD data, the value of *Out* will be the same regardless of whether the value of *Num* is USINT#1 or USINT#17.

Conversion Instructions

Instruction	Name	Page	Instruction	Name	Page
Swap	Swap Bytes	2-404	StringToFixNum	Text String-to-Fixed-decimal Conversion	2-453
Neg	Reverse Sign	2-405	DtToString	Date and Time-to-Text String Conversion	2-456
Decoder	Bit Decoder	2-407	DateToString	Date-to-Text String Conversion	2-458
Encoder	Bit Encoder	2-410	TodToString	Time of Day-to-Text String Conversion	2-459
BitCnt	Bit Counter	2-412	GrayToBin_** and BinToGray_**	Gray Code-to-Binary Code Conversion Group/ Binary Code-to-Gray Code Conversion	2-461
ColmToLine_**	Column to Line Conversion Group	2-413	StringToAry	Text String-to-Array Conversion	2-463
LineToColm	Line to Column Conversion	2-415	AryToString	Array-to-Text String Conversion	2-465
Gray	Gray Code Conversion	2-417	DispartDigit	Four-bit Separation	2-467
UTF8ToSJIS	UTF-8 to SJIS Character Code Conversion	2-422	UniteDigit_**	Four-bit Join Group	2-469
SJISToUTF8	SJIS to UTF-8 Character Code Conversion	2-424	Dispart8Bit	Byte Data Separation	2-471
PWLApprox and PWLApproxNo-LineChk	Broken Line Approximation with Broken Line Data Check and Bro-ken Line Approximation without Broken Line Data Check	2-426	Unite8Bit_**	Byte Data Join Group	2-473
PWLLineChk	Broken Line Data Check	2-432	ToAryByte	Conversion to Byte Array	2-475
MovingAverage	Moving Average	2-435	AryByteTo	Conversion from Byte Array	2-480
DispartReal	Separate Mantissa and Exponent	2-441	SizeOfAry	Get Number of Array Elements	2-485
UniteReal	Combine Real Number Mantissa and Exponent	2-444	PackWord	2-byte Join	2-487
NumToDecString and NumToHex-String	Fixed-length Decimal Text String Conversion/ Fixed-length Hexadecimal Text String Conversion	2-446	PackDword	4-byte Join	2-489
HexStringToNum_**	Hexadecimal Text String-to-Number Conversion Group	2-449	LOW-ER_BOUND/UPPER_BOUND	Get First Number of Array/ Get Last Number of Array	2-491
FixNumToString	Fixed-decimal Number-to-Text String Conversion	2-451			

Swap

The Swap instruction swaps the upper byte and lower byte of a 16-bit value.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Swap	Swap Bytes	FUN		Out:=Swap(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	Depends on data type.	---	0
Out	Conversion result	Output	Conversion result	Depends on data type.	---	---

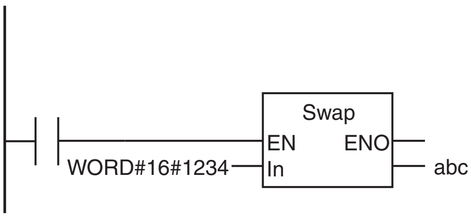
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In			OK																	
Out			OK																	

Function

The Swap instruction swaps the upper byte and lower byte of data to convert *In* and assigns the result to conversion result *Out*.

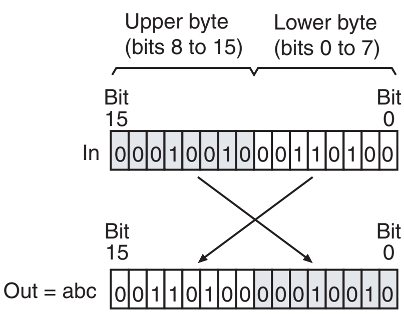
The following example is for when *In* is WORD#16#1234.

LD



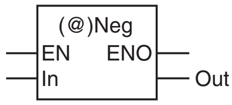
ST

```
abc:=Swap(WORD#16#1234);
```



Neg

The Neg instruction reverses the sign of a number.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Neg	Reverse Sign	FUN		Out:=Neg(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	Depends on data type.	---	*
Out	Conversion result	Output	Conversion result	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Out						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					

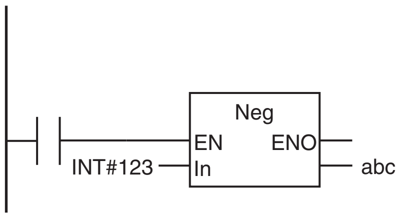
Function

The Neg instruction reverses the sign of data to convert *In*. The value of *Out* depends on the data type of *In*.

Data type of *In*	Value of *Out*
Signed integer: SINT, INT, DINT, or LINT	All bits in *In* are reversed and then 1 is added. (This is the same as multiplying *In* by –1.)
Unsigned integers: USINT, UNIT, UDINT, or ULINT	All bits in *In* are reversed and then 1 is added.
Real numbers: REAL or LREAL	*In* × (–1)

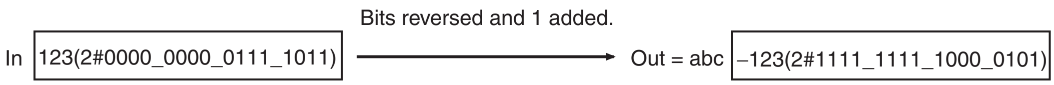
The following example is for when *In* is INT#123.

LD

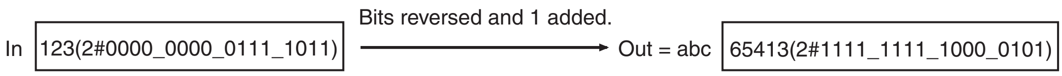


ST

abc:=Neg(INT#123);

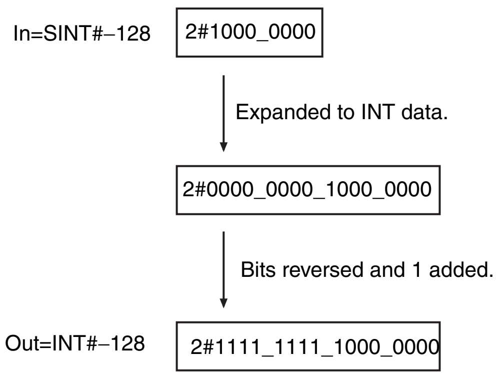


The following example is for when *In* is UINT#123.



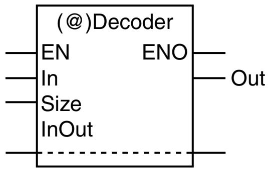
Precautions for Correct Use

If you use a different data type for *In* and *Out*, make sure the valid range of *Out* includes the valid range of *In.* Otherwise, an error will not occur and the value of *Out* will be an illegal value. For example, if the value of *In* is SINT#-128 and the data type of *Out* is INT, the value of *Out* will be INT#-128 instead of INT#128.



Decoder

The Decoder instruction sets the specified bit to TRUE and the other bits to FALSE in array elements that consist of a maximum of 256 bits.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Decoder	Bit Decoder	FUN		Decoder(In, Size, InOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Conversion bit position	Input	Bit position to convert	Depends on data type.	---	0
Size	Bits to convert		Number of bits to convert	0 to 8	Bits	1
InOut[] (array)	Array to convert	In-out	Array to convert	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK																		
Size						OK														
InOut[] (array)	OK	OK	OK	OK	OK															
Out	OK																			

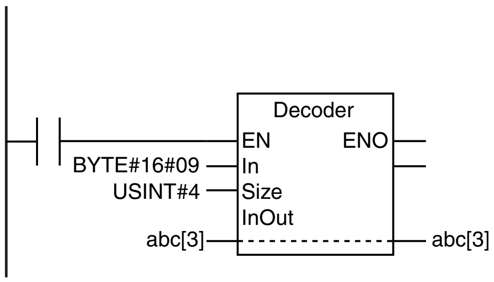
Function

The Decoder instruction converts array elements with 2^{Size} bits that start from *InOut[0]* in array to convert *InOut[]*. It sets the specified bit to TRUE. It sets the other bits to FALSE. The bit to make TRUE is specified by the *Size* bits in the lower byte of conversion bit position *In*. Always attach the element number to the in-out parameter that is passed to *InOut[]*, e.g., *array[3]*.

Consider an example where *In* is BYTE#16#09, *Size* is USINT#4, and *InOut[]* is a BYTE array. The value of the *Size* bits in the lower bits of *In* is 16#9, which is 9 decimal. Therefore, the ninth bit from the least-significant bit of *InOut[]* is made TRUE and the other bits are made FALSE.

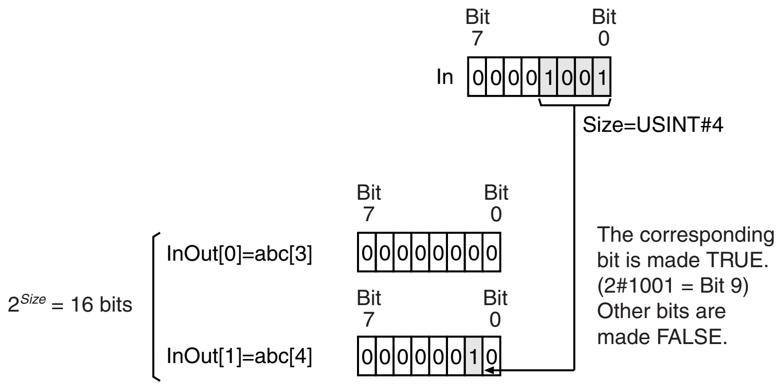
InOut[] is a BYTE array, so the ninth bit from the least-significant bit is bit 1 in *InOut[1]*. Therefore, bit 1 in *InOut[1]* is made TRUE, all other bits in *InOut[1]* are made FALSE, and all bits in *InOut[0]* are made FALSE.

LD

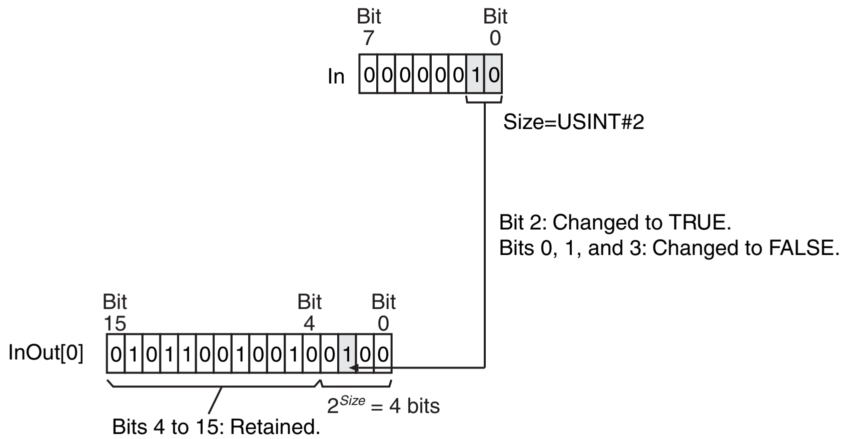


ST

```
Decoder(BYTE#16#09, USINT#4, abc[3]);
```



If the number of bits in the elements of *InOut[]* is larger than the number of bits specified by *Size*, the values of the remaining bits are retained. Consider an example where *In* is BYTE#16#02, *Size* is USINT#2, and *InOut[]* is a WORD array.
Size is USINT#2, so the value is set in the lower 4 bits of *InOut[0]*. The values of the remaining bits in *InOut[0]* (bits 4 to 15) are retained.



Additional Information

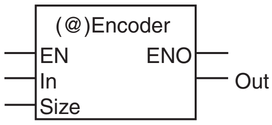
Use the Encoder instruction (page 2-410) to find the position of the highest TRUE bit in array elements that consist of a maximum of 256 bits.

Precautions for Correct Use

- If the value of *Size* is 0, all bits in *InOut[]* change to FALSE.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *InOut[]* will not change.
  - The value of *Size* is outside of the valid range.
  - The value of 2^{Size} exceeds the number of bits in the array elements of *InOut[]*.

Encoder

The Encoder instruction finds the position of the highest TRUE bit in array elements that consist of a maximum of 256 bits.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Encoder	Bit Encoder	FUN		Out:=Encoder(In, Size);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array to convert	Input	Array to convert	Depends on data type.	---	*
Size	Bits to convert		Number of bits to convert	0 to 8	Bits	1
Out	Conversion result	Output	Conversion result	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)	OK	OK	OK	OK	OK															
Size						OK														
Out		OK																		

Function

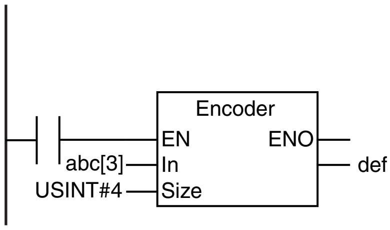
The Encoder instruction finds the position of a TRUE bit in a specified range of bits in array to convert *In[]*. The instruction looks for a TRUE bit in 2^{Size} bits from *In[0]*. The position of the TRUE bit in this range is expressed in binary and stored in the *Size* bits in the lower bits of conversion result *Out*. FALSE is stored in the remaining bits of *Out*.

If there is more than one TRUE bit in the specified range, the position of the highest bit that is TRUE is found. Always attach the element number to input parameter that is passed to *In[]*, e.g., *array[3]*.

Consider an example for when *Size* is USINT#4 and *In[]* is a BYTE array. *Size* is USINT#4, so the range in which to find a TRUE bit is 2^4, or 16 bits, from *In[0]*. In the following diagram, the ninth bit in the range is TRUE.

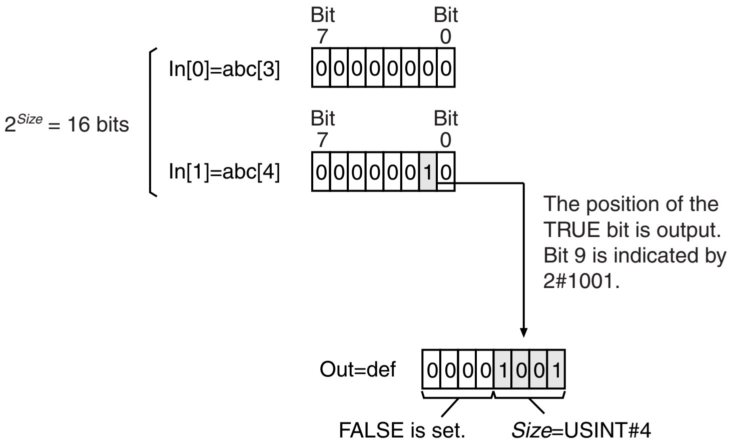
Size is USINT#4, so 2#1001 (i.e., 9) is stored in the lower 4 bits of *Out*. FALSE is stored in the upper four bits of *Out*.

LD



ST

def:=Encoder(abc[3], USINT#4);



Additional Information

Use the Decoder instruction (page 2-407) to make one bit TRUE and the other bits FALSE in array elements that consist of a maximum of 256 bits.

Precautions for Correct Use

- If the value of *Size* is 0, all bits in *Out* change to FALSE.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of *Size* is outside of the valid range.
  - The value of 2^{Size} exceeds the number of bits in the array elements of *In[]*.
  - The value of all bits in *In[]* that are specified by *Size* change to FALSE.

BitCnt

The BitCnt instruction counts the number of TRUE bits in a bit string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
BitCnt	Bit Counter	FUN	(@)BitCnt EN ENO In Out	Out:=BitCnt(In);

Variables

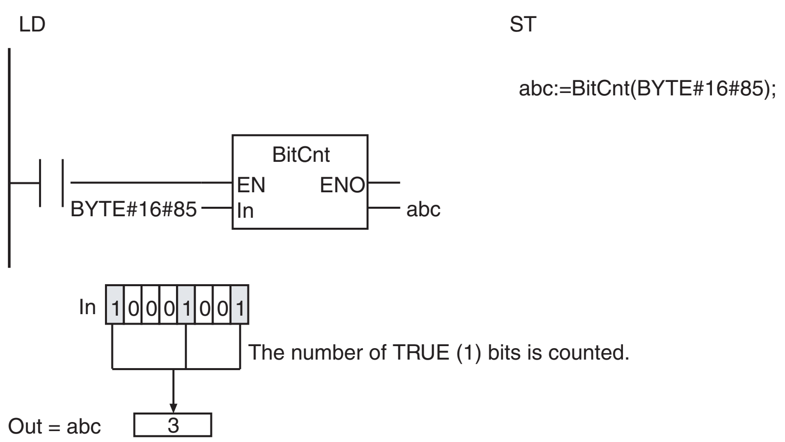
Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Count string	Input	String in which to count TRUE bits	Depends on data type.	---	*
Out	Count result	Output	Number of TRUE bits	0 to No. of bits in *In*	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Out						OK														

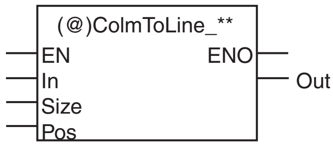
Function

The BitCnt instruction counts the number of TRUE bits in count string *In*. The following example is for when *In* is BYTE data with a value of BYTE#16#85.



ColmToLine_**

The ColmToLine_** instruction extracts bit values from the specified position of array elements and outputs them as a bit string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ColmToLine_**	Column to Line Conversion Group	FUN	 "**" must be a bit string data type.	Out:=ColmToLine_**(In, Size, Pos); "**" must be a bit string data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array to convert	Input	Array to convert	Depends on data type.	---	*
Size	Number of elements to convert		Number of elements in *In[]* to convert	0 to No. of bits in *Out*		1
Pos	Bit position to convert		Bit position to convert	0 to No. of bits in *In[]* – 1		0
Out	Conversion result	Output	Conversion result	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)		OK	OK	OK	OK															
Size						OK														
Pos						OK														
Out		OK	OK	OK	OK															

Function

The ColmToLine_** instruction extracts bit values from the specified position of array elements and outputs them in order as a bit string.

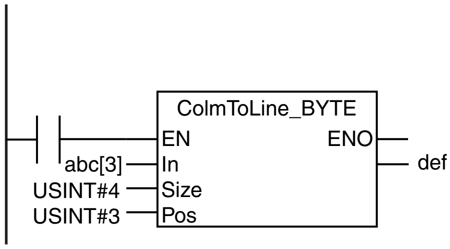
First, *Size* elements are extracted from array to convert *In[]* starting from *In[0]*. Then, only the values of bits in *Pos* are extracted. These are placed in order in a bit string of *Size* bits and stored in conversion result *Out* from the least-significant bit. FALSE is stored in the remaining bits of *Out*.

The name of the instruction is determined by the data type of *Out*. For example, if *Out* is the BYTE data type, the instruction is ColmToLine_BYTE.

Always attach the element number to input parameter that is passed to *In[]*, e.g., *array[3]*.

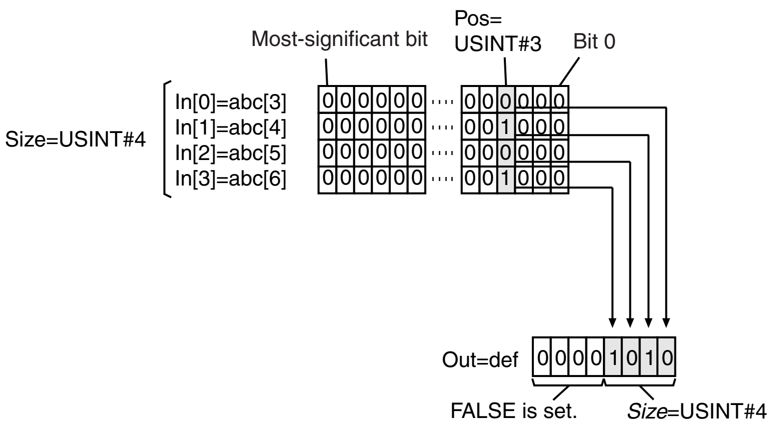
The following example shows the ColmToLine_BYTE instruction when *Pos* is USINT#3 and *Size* is USINT#4.

LD



ST

```
def:=ColmToLine_BYTE(abc[3], USINT#4, USINT#3);
```



Additional Information

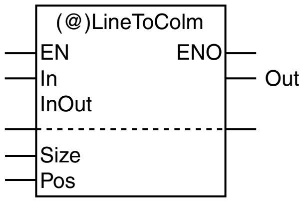
Use the LineToColm instruction (page 2-415) to output a bit string to the specified bit position in array elements.

Precautions for Correct Use

- If the value of *Size* is 0, all bits in *Out* change to FALSE.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of *Size* is outside of the valid range.
  - The value of *Pos* is outside of the valid range.
  - The value of *Size* exceeds the array area of *In[]*.

LineToColm

The LineToColm instruction takes the bits from a bit string and outputs them to the specified bit position in array elements.

Instruction	Name	FB/FUN	Graphic expression	ST expression
LineToColm	Line to Column Conversion	FUN		LineToColm(In, InOut, Size, Pos);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert		Data to convert	Depends on data type.		*
Size	Number of elements in result	Input	Number of elements in result	0 to No. of bits in *In*	---	1
Pos	Conversion bit position		Bit position to receive the conversion	0 to No. of bits in *InOut[]* – 1		0
InOut[] (array)	Conversion result array	In-out	Conversion result	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Size						OK														
Pos						OK														
InOut[] (array)		OK	OK	OK	OK															
Out	OK																			

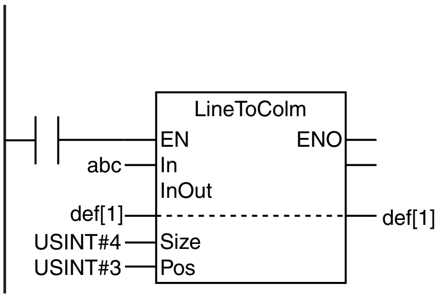
Function

The LineToColm instruction takes the bits from a bit string and outputs them to the specified bit position in array elements.

First, *Size* bits are extracted from the least-significant bit of data to convert *In*. These bits are treated individually. Then, the bits are stored in conversion result array *InOut[]* in the *Pos* bit of the elements starting from *InOut[0]. Size* specifies the number of array elements to receive bits. The values of all bits for which values are not stored are retained.

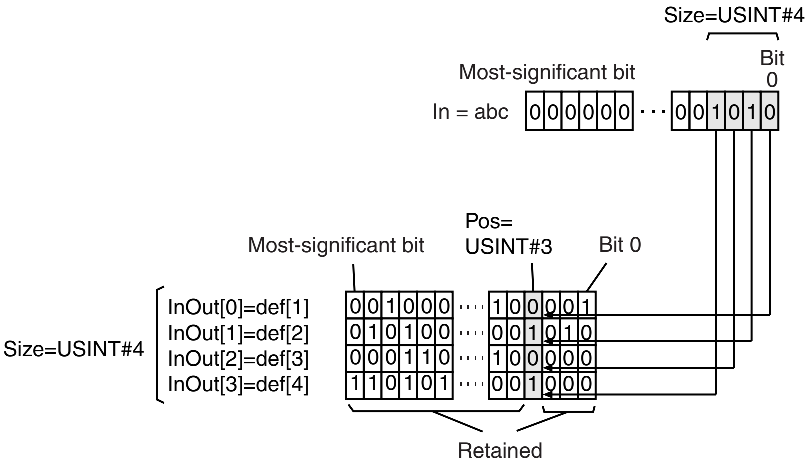
The following example is for when *Pos* is USINT#3 and *Size* is USINT#4.

LD



ST

```
LineToColm(abc, def[1], USINT#4, USINT#3);
```



Additional Information

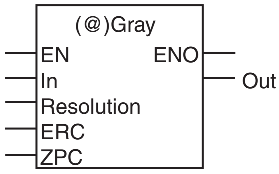
Use the ColmToLine_** instruction (page 2-413) to extract bit values from the specified position of array elements and output them as a bit string.

Precautions for Correct Use

- If the value of *Size* is 0, the values in *InOut[]* do not change.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *InOut[]* will not change.
  - The value of *Size* is outside of the valid range.
  - The value of *Pos* is outside of the valid range.
  - The value of *Size* exceeds the array area of *InOut[]*.

Gray

The Gray instruction converts a gray code into an angle.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Gray	Gray Code Conversion	FUN		Out:=Gray(In, Resolution, ERC, ZPC);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Gray code to convert	Depends on data type.	---	0
Resolution	Resolution		Resolution	_R256, _R1B to _R15B, _R360, _R720, or _R1024		_R256
ERC	Encoder remainder correction		Encoder remainder correction	0 to *Resolution*		0
ZPC	Zero point correction		Zero point correction			
Out	Conversion result	Output	Conversion result	*	°	---

* 0 to 3.59999999999999e+2

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In			OK																	
Resolution	Refer to *Function* for the enumerators of the enumerated type _eGRY_RESOLUTION.																			
ERC							OK													
ZPC							OK													
Out															OK					

Function

The Gray instruction converts the gray code in *In* (the output value from a rotary encoder) to an angle. The conversion result *Out* is in degrees.

The data type of *Resolution* is enumerated type _eGRY_RESOLUTION. The meaning of the enumerators are as follows:

Enumerator	Meaning
_R256	256
_R1B	1-bit (2)
_R2B	2-bit (4)
_R3B	3-bit (8)
_R4B	4-bit (16)
_R5B	5-bit (32)
_R6B	6-bit (64)
_R7B	7-bit (128)
_R8B	8-bit (256)
_R9B	9-bit (512)
_R10B	10-bit (1024)
_R11B	11-bit (2048)
_R12B	12-bit (4096)
_R13B	13-bit (8192)
_R14B	14-bit (16384)
_R15B	15-bit (32768)
_R360	360
_R720	720
_R1024	1024

Gray Code

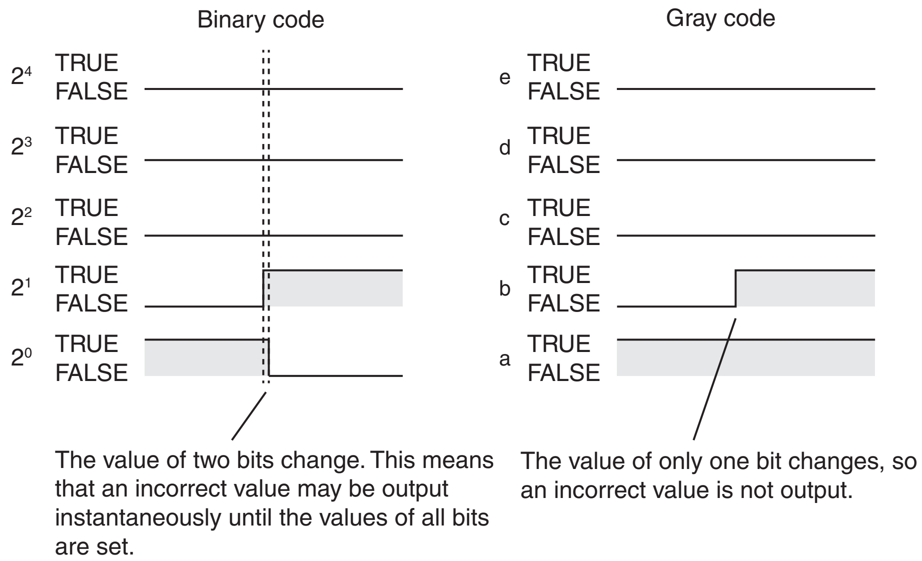
The Gray code is a reflected binary code. Two successive values, such as 0 and 1 and 1 and 2, differ in only one bit. Gray codes are used for the output from absolute encoders.

The following tables shows the 4-bit Binary code and Gray code.

Decimal number	Binary code				Gray code			
	2^3	2^2	2^1	2^0	d	c	b	a
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	0

Using the Gray code enables prevention of instantaneously incorrect output values because only one bit in the Gray code will change when the output value of the encoder is incremented or decremented by 1. The following figure shows the difference in the output value from an encoder for the Gray code and Binary code.

Difference When Output Value Changes from 1 to 2



ERC: Encoder Remainder Correction

The *ERC* variable is used to specify the Gray code range when the encoder resolution is not a power of 2. The range is specified so that there is only one bit difference between the maximum and minimum encoder output values. For example, consider the use of an absolute encoder with a resolution of 360. Nine bits are used for the Gray code. The range that can be expressed with nine bits is 0 to 511. In this case, a range of 180 from the center of 0 to 511 is used for the Gray code, i.e., 76 to 435. Therefore, a Gray code of 001101010 (76 decimal) is output for an output value of 0, and a Gray code of 101101010 (435 decimal) is output for an output value of 359. There is a difference in only one bit between these values. In this case, the value of encoder remainder correction *ERC* is 76.

	Decimal	Gray code								
		i	h	g	f	e	d	c	b	a
	0	0	0	0	0	0	0	0	0	0
	...	...								
For an output value of 0	76	0	0	1	1	0	1	0	1	0
	...	...								
	255	0	1	0	0	0	0	0	0	0
	256	1	1	0	0	0	0	0	0	0
	...	...								
For an output value of 359	435	1	0	1	1	0	1	0	1	0
	...	...								
	511	1	0	0	0	0	0	0	0	0

There is a difference in only one bit.

ERC (encoder remainder correction): 76

180

180

76

Resolution: 360

ZPC: Zero Point Correction

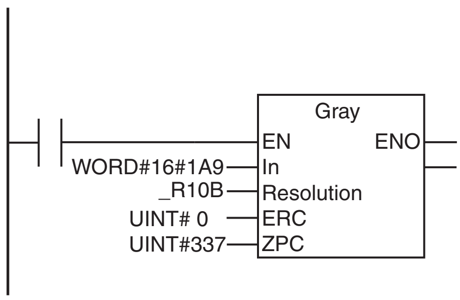
The *ZPC* variable is set to offset the zero position of the rotary encoder. For example, to offset the zero position for a rotary encoder with a resolution of 256, the value of *ZPC* would be 256 × (90/360), or 64.

Notation Example

The following example is for when *In* is WORD#16#1A9, *Resolution* is _R10B, *ERC* is UINT#0, and *ZPC* is UINT#337.

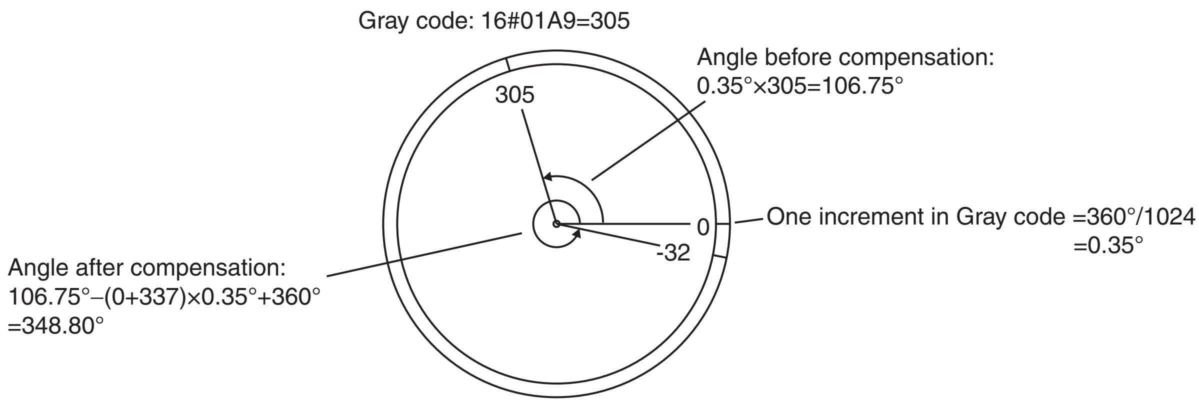
First, the resolution is 10 bits, so one increment in the Gray code is 360°/1,024, or 0.35°. A decimal value of 305 corresponds to a Gray code of 16#01A9. Therefore, the angle before compensation is 0.35° × 305, or 106.75°. The value of *ERC* is 0 and the value of *ZPC* is 377. Therefore, the angle after compensation is 106.75° − (0 + 337) × 0.35°, or −11.20°. The range of the values of *Out* is 0 or greater, so the value is −11.20° + 360°, or 348.80°. The value of *Out* will be LREAL#348.8.

LD



ST

```
abc:=Gray(WORD#16#1A9, _R10B, UINT#0, UINT#337);
```



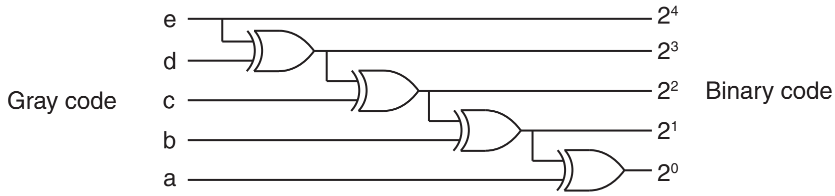
Additional Information

Refer to the user documentation for your rotary encoder for the values to specify for *Resolution* and *ERC*.

Converting from Gray Code to Binary Code

The following processing can be used to convert from Gray code to Binary code. The logic symbols in the figure represent logical exclusive ORs.

Conversion Circuit for Five Bits



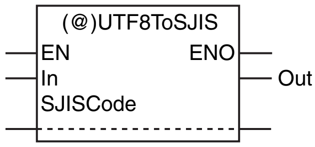
Precautions for Correct Use

An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.

- The value of *Resolution* is outside of the valid range.
- The value of *ERC* exceeds the resolution that is specified in *Resolution*.
- The value of Z*PC* exceeds the resolution that is specified in *Resolution*.
- *In*, when converted to a bit string, is smaller than the value of *ERC*.
- The value of the bit string after correction for *ERC* exceeds the resolution that is specified in *Resolution*.

UTF8ToSJIS

The UTF8ToSJIS instruction converts a UTF-8 text string to a SJIS BYTE array.

Instruction	Name	FB/FUN	Graphic expression	ST expression
UTF8ToSJIS	UTF-8 to SJIS Character Code Conversion	FUN		Out:=UTF8ToSJIS(In, SJISCode);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Text string to convert	Input	Text string to convert	Depends on data type.	---	''
SJIS-Code[] (array)	SJIS array	In-out	Array of SJIS character codes	Depends on data type.	---	---
Out	Number of converted elements	Output	Number of elements stored in *SJISCode[]*	0 to 1985	---	---

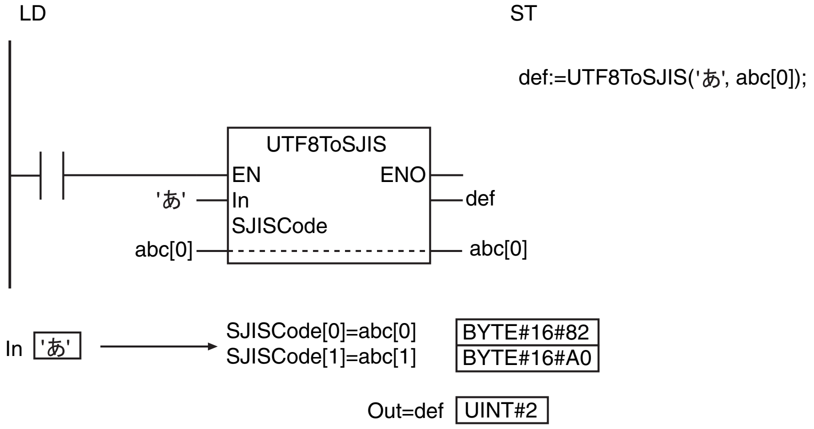
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
SJIS-Code[] (array)		OK																		
Out							OK													

Function

The UTF8ToSJIS instruction converts the text string to convert in *In* (a UTF-8 text string) to a SJIS array in *SJISCode[]* (a BYTE array of SJIS character codes). Each byte of the converted data is stored in order starting from *SJISCode[0]*.

The number of elements of converted data that was stored in *SJISCode[]* is stored as the number of converted elements in *Out*.

The following example is for when *In* is 'あ'.



Precautions for Correct Use

- NULL characters at the end of *In* are not converted. They are also not counted for the number of converted elements.
- If the *In* text string contains only the NULL character, the value of *Out* will be 0 and *SJISCode[]* will not change.
- The elements of *SJISCode[]* past the number of elements specified in *Out* do not change. For example, if the number of converted elements is 5, *SJISCode[5]* and later elements do not change.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* and *SJISCode[]* will not change.
  - The number of elements in the conversion result exceeds the size of the output parameter that is connected to *SJISCode[]*.
  - The contents of *In* includes characters that cannot be converted.

Version Information

A CPU Unit with unit version 1.01 or later and Sysmac Studio version 1.02 or higher are required to use this instruction.

SJISToUTF8

The SJISToUTF8 instruction converts a SJIS BYTE array to a UTF-8 text string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SJISToUTF8	SJIS to UTF-8 Character Code Conversion	FUN	(@)SJISToUTF8 EN ENO In Out Size	Out:=SJISToUTF8(In, Size);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	SJIS array to convert	Input	Array of SJIS character codes to convert*1	Depends on data type.	---	*2
Size	Number of SJIS array elements		Number of elements of *In[]* to convert			---
Out	Resulting text string	Output	UTF-8 text string after conversion	Depends on data type.	---	---

*1 The maximum number of elements is 1,986, including the NULL character (BYTE#16#00). The maximum number of elements is 1,985 without the NULL character.

*2 If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)		OK																		
Size							OK													
Out																				OK

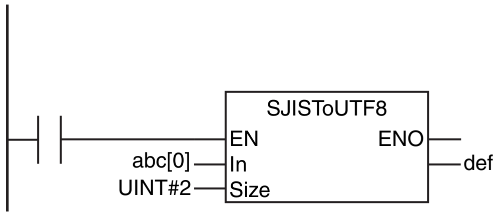
Function

The SJISToUTF8 instruction converts the elements in a SJIS array to convert in *In[]* (a BYTE array) to a UTF-8 text string. *Size* number of elements are converted starting with *In[0]*. However, if there is a NULL character (BYTE#16#00) before *Size* elements are converted, conversion is canceled at that point.

The resulting text string after conversion is stored as resulting text string in *Out*. A NULL character is placed at the end of *Out*.

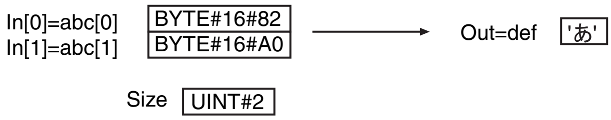
The following example is for when *In[0]* is BYTE#16#82, *In[1]* is BYTE#16#A0, and *Size* is UINT#2.

LD



ST

```
def:=SJISToUTF8(abc[0], UINT#2);
```



Precautions for Correct Use

- If the value of *Size* is 0, *Out* is a text string containing only the NULL character.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of Size exceeds the number of elements in *In[]*.
  - The contents of *In[]* includes characters that cannot be converted.

Version Information

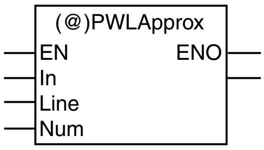
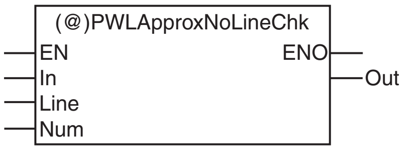
A CPU Unit with unit version 1.01 or later and Sysmac Studio version 1.02 or higher are required to use this instruction.

PWLApprox and PWLApproxNoLineChk

The PWLApprox and PWLApproxNoLineChk instructions perform broken line approximations for integer or real number data.

PWLApprox: Checks to see if the broken line data is valid.

PWLApproxNoLineChk: Does not check to see if the broken line data is valid.

Instruction	Name	FB/FUN	Graphic expression	ST expression
PWLApprox	Broken Line Approximation with Broken Line Data Check	FUN		Out:=PWLApprox(In, Line, Num);
PWL Approx NoLineChk	Broken Line Approximation without Broken Line Data Check	FUN		Out:=PWLApproxNoLineChk(In, Line, Num);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	Depends on data type.	---	*
Line[] (array)	Broken line data array		Broken line data array			
Num	Number of broken line data		Number of broken line data			1
Out	Conversion result	Output	Conversion result	Depends on data type.	---	---

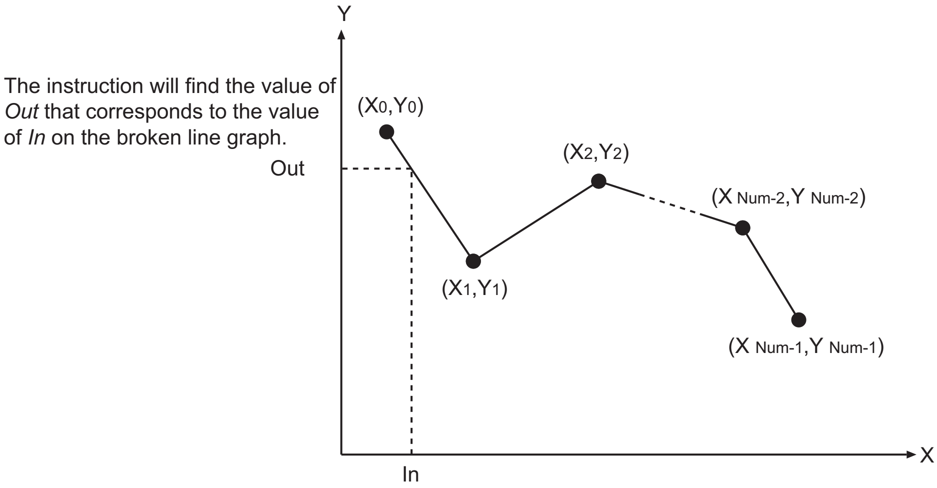
* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Line[] (array)	Must be an array with elements that have the same data type as *In*.																			
Num							OK													
Out						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					

Function

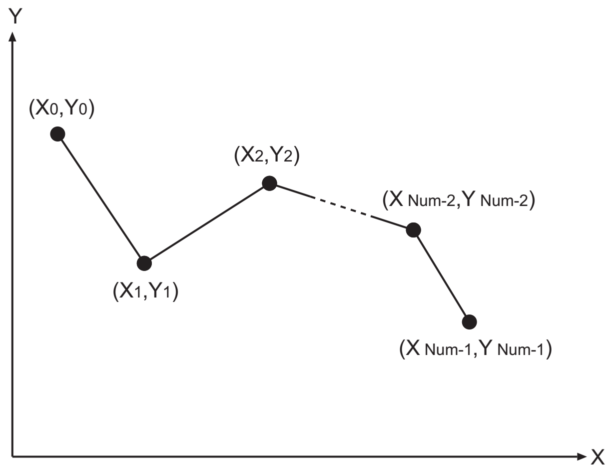
The PWLApprox and PWLApproxNoLineChk instructions perform approximation for data to convert *In*. The approximation is based on broken line data that consists of *Num* times 2 elements that start with *Line[0,0]* in broken line data array *Line[]*.

As shown below, the Y coordinate that corresponds to the X coordinate *In* of the broken line data is assigned to conversion result *Out*.



Elements of Broken Line Data Array *Line[]* and Number of Broken Line Data *Num*

Line[] must be a two-dimensional or three-dimensional array. Set the number of elements for the first dimension to 2. Then use the coordinate values (X_0,Y_0), (X_1,Y_1), etc., of the points in the broken line data as the elements of *Line[]* as shown in the following figure. The number of broken line data *Num* is one half of the number of elements of *Line[]*, which is used in the broken line approximation calculations.



Using a Two-dimensional Array for *Line[]*

Line[0,0]	X_0
Line[0,1]	Y_0
Line[1,0]	X_1
Line[1,1]	Y_1
Line[2,0]	X_2
Line[2,1]	Y_2
:	:
Line[Num-1,0]	X_{Num-1}
Line[Num-1,1]	Y_{Num-1}

Using a Three-dimensional Array for *Line[]*

Line[0,0,0]	X_0
Line[0,0,1]	Y_0
Line[0,1,0]	X_1
Line[0,1,1]	Y_1
Line[0,2,0]	X_2
Line[0,2,1]	Y_2
:	:
Line[0, Num-1,0]	X_{Num-1}
Line[0, Num-1,1]	Y_{Num-1}

Notation Example

An example of approximation when the value of *In* is LREAL#3.0 for broken line data array *abc[]* with four elements is given below. The values of the elements of *abc[]* are given below for when *Num* is UINT#4.

abc[0.0] = X_0 = LREAL#1.0, abc[0,1] =Y_0 = LREAL#5.0,

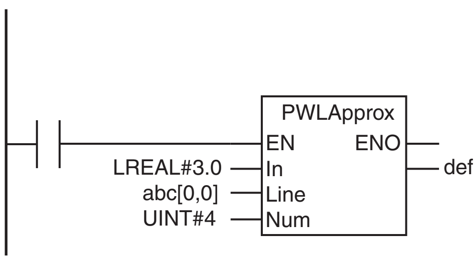
abc[1.0] = X_1 = LREAL#2.0, abc[1,1] =Y_1 = LREAL#6.0,

abc[2.0] = X_2 = LREAL#4.0, abc[2,1] =Y_2 = LREAL#2.0,

abc[3.0] = X_3 = LREAL#5.0, abc[3,1] =Y_3 = LREAL#3.0

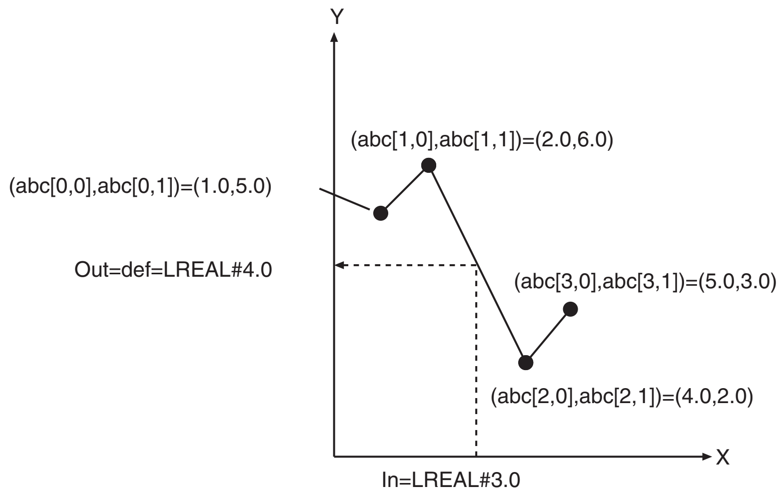
The value of conversion result *Out* will be LREAL#4.0.

LD



ST

```
def:=PWLApprox(LREAL#3.0, abc[0,0], UINT#4);
```



Difference between the PWLApprox and PWLApproxNoLineChk Instructions

The PWLApprox and PWLApproxNoLineChk instructions are different in whether the validity of *In* and *Line[]* are checked. This also makes the processing times different. The specifications of both instructions are given in the following table.

Instruction	Checks	Processing when the data is not valid	Processing time
PWLApprox	• The contents of *Line[]* are checked to make sure the elements are in ascending order of the X coordinates. • If *In* and *Line[]* are integers, *In* and the elements of *Line[]* are checked to make sure they are not nonnumeric data, positive infinity, or negative infinity.	• An error occurs. • The value of *ENO* will be FALSE. • The value of *Out* will not change.	Long
PWLApprox NoLineChk	No checks are performed.	• An error will not occur. • The value of *ENO* will be TRUE. • A valid value may not be output to *Out*.	Short

PWLApproxNoLineChk and PWLLineChk Instructions

Although the PWLApproxNoLineChk instruction does not check the validity of *In* and *Line[]*, the processing time is short. Therefore, if you can be sure that the input variables are valid, it is better to use the PWLApproxNoLineChk instruction rather than the PWLApprox instruction.

The PWLLineChk instruction (page 2-432) checks the contents of *Line[]* to see if the X coordinates are in ascending order. Therefore, you can shorten the processing time if you normally use the PWLApproxNoLineChk instruction and combine the PWLLineChk instruction with it only when you cannot ensure that the X coordinates in *Line[]* are in ascending order.

Additional Information

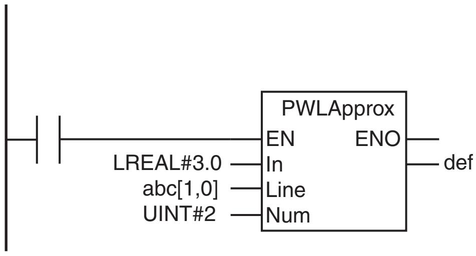
You can also shorten the processing time by restricting the range of elements in the broken line data array that is used for approximation conversion.

In the previous example, the processing time will be shorter for the value of *In* (LREAL#3.0) if the values of the X coordinates in the elements of the broken line data array consist of only the four elements that are close to 3.0 (abc[1,0],abc[1,1])=(2.0,6.0) and (abc[2,0],abc[2,1])=(4.0,2.0).

In this case, *Num* is UINT#2 and the element of *abc[]* that is passed to *Line[]* is *abc[1,0]*.

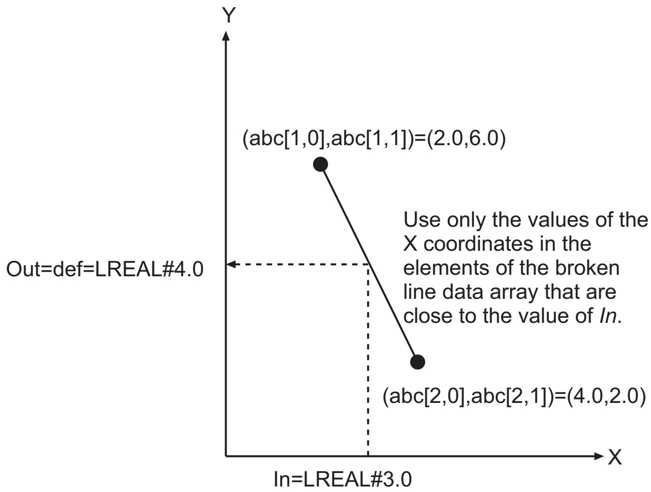
The conversion result *Out* is still LREAL#4.0.

LD



ST

```
def:=PWLApprox(LREAL#3.0, abc[1,0], UINT#2);
```



Precautions for Correct Use

- If the value of *In* is smaller than the value of *Line[0,0]* (i.e., the value of X_1), then the value of *Out* will be the value of *Line[0,1]* (i.e., the value of Y_1).
- If the value of *In* is larger than the value of *Line[Num–1,0]* (i.e., the value of X_{Num}), then the value of *Out* will be the value of *Line[Num–1,1]* (i.e., the value of Y_{Num}).
- *Line[]* must be a two-dimensional or three-dimensional array. Set the number of elements for the first dimension to 2.
- If the value of *Num* is 0, the value of *Out* is 0.
- An error will occur for the PWLApprox instruction in the following cases. *ENO* will be FALSE, and *Out* will not change. An error will not occur in these cases for the PWLApproxNoLineChk instruction.
  - The X coordinates of the broken line data are not in ascending order, the condition $X_1 < X_2 < ... < X_{Num}$ is not met.
  - *In* and *Line[]* are REAL data and their values are nonnumeric data, positive infinity, or negative infinity.
- An error will occur for the PWLApprox instruction and the PWLApproxNoLineChk instruction in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of *Num* exceeds the array area of *Line[]*.
  - The value of *In* exceeds the X coordinates in the broken line data that is specified for *Line[]*.

Version Information

A CPU Unit with unit version 1.03 or later and Sysmac Studio version 1.04 or higher are required to use the PWLApproxNoLineChk instruction.

PWLLineChk

The PWLLineChk instruction is used to check whether the X coordinates in the broken line data that is used for a Broken Line Approximation without Broken Line Data Check instruction are in ascending order.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
PWLLineChk	Broken Line Data Check	FUN	(@)PWLLineChk EN — Out Line Num	Out:=PWLLineChk(Line, Num);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Line[] (array)	Broken line data array	Input	Broken line data array	Depends on data type.	---	(*)
Num	Number of broken line data		Number of broken line data			1
Out	Result	Output	Result	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

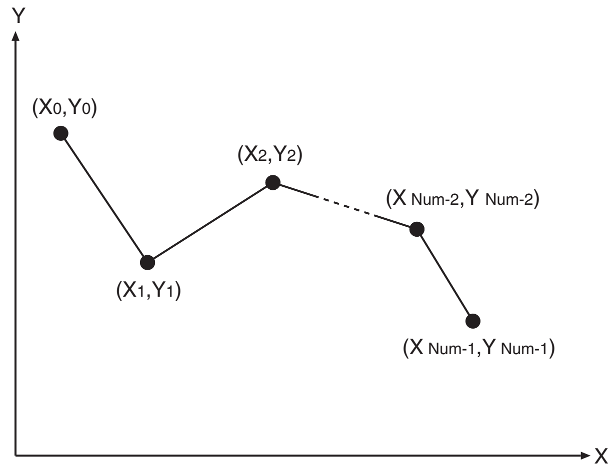
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Line[] (array)						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Num							OK													
Out	OK																			

Function

The PWLLineChk instruction is used to check whether the X coordinates in the broken line data array *Line[]* that is used for a Broken Line Approximation without Broken Line Data Check (PWLApproxNo-LineChk) instruction are in ascending order. If the X coordinates are in ascending order, result *Out* will be TRUE. If they are not, result *Out* will be FALSE.

Elements of Broken Line Data Array *Line[]* and Number of Broken Line Data *Num*

Line[] must be a two-dimensional or three-dimensional array. Set the number of elements for the first dimension to 2. Then use the coordinate values (X_0,Y_0), (X_1,Y_1), etc., of the points in the broken line data as the elements of *Line[]* as shown in the following figure. The number of broken line data *Num* is one half of the number of elements of *Line[]*, which is used in the broken line approximation calculations.



Using a Two-dimensional Array for *Line[]*

Line[0,0]	X_0
Line[0,1]	Y_0
Line[1,0]	X_1
Line[1,1]	Y_1
Line[2,0]	X_2
Line[2,1]	Y_2
:	:
Line[Num-1,0]	X_{Num-1}
Line[Num-1,1]	Y_{Num-1}

Using a Three-dimensional Array for *Line[]*

Line[0,0,0]	X_0
Line[0,0,1]	Y_0
Line[0,1,0]	X_1
Line[0,1,1]	Y_1
Line[0,2,0]	X_2
Line[0,2,1]	Y_2
:	:
Line[0, Num-1,0]	X_{Num-1}
Line[0, Num-1,1]	Y_{Num-1}

Notation Example

An example of determining whether the X coordinates are in ascending order in the broken line data array *abc[]* with four elements is given below. The values of the elements of *abc[]* are given below for when *Num* is UINT#4.

abc[0.0] = X_0 = LREAL#1.0, abc[0,1] = Y_0 = LREAL#5.0,

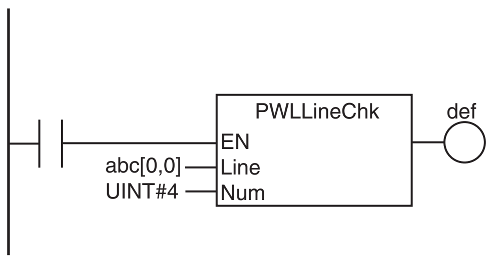
abc[1.0] = X_1 = LREAL#6.0, abc[1,1] = Y_1 = LREAL#6.0,

abc[2.0] = X_2 = LREAL#4.0, abc[2,1] = Y_2 = LREAL#2.0,

abc[3.0] = X_3 = LREAL#5.0, abc[3,1] = Y_3 = LREAL#3.0

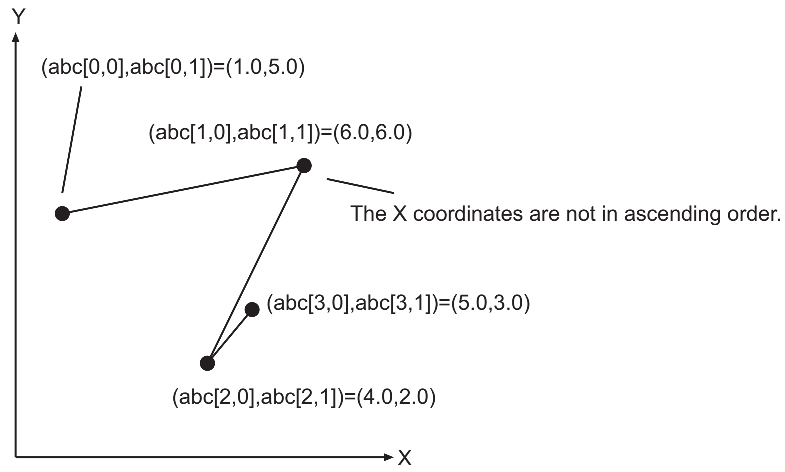
The X coordinates are not in ascending order, so the value of *Out* is FALSE.

LD



ST

```
def:=PWLLineChk(abc[0,0], UINT#4);
```



Additional Information

- Use the PWLLineChk in combination with the PWLApproxNoLineChk instruction. Refer to *PWLApprox and PWLApproxNoLineChk Instructions* (page 2-426) for details on the PWLApproxNoLineChk instruction.
- Use the PWLApprox instruction to check the broken line data every time you perform broken line approximation. Refer to *PWLApprox and PWLApproxNoLineChk Instructions* (page 2-426) for details on the PWLApprox instruction. The processing time of the PWLApproxNoLineChk instruction is shorter than the processing time of the PWLApprox instruction.

Precautions for Correct Use

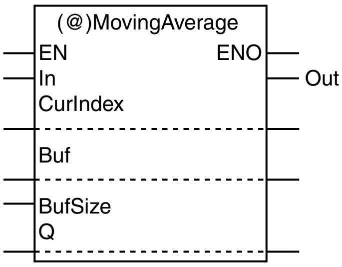
- *Line[]* must be a two-dimensional or three-dimensional array. Set the number of elements for the first dimension to 2.
- An error will occur in the following cases. *Out* will be FALSE.
  - The value of *Num* exceeds the array area of *Line[]*.
  - *Line[]* is REAL data and an element is nonnumeric data, positive infinity, or negative infinity.

Version Information

A CPU Unit with unit version 1.03 or later and Sysmac Studio version 1.04 or higher are required to use this instruction.

MovingAverage

The MovingAverage instruction calculates a moving average.

Instruction	Name	FB/FUN	Graphic expression	ST expression
MovingAverage	Moving Average	FUN		Out:=MovingAverage(In, CurIndex, Buf, BufSize, Q);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Input value	Input	Number to include in average	Depends on data type.	---	*
BufSize	Maximum number stored	Input	Maximum number of elements to include in average	Depends on data type.	---	1
CurIndex	Input value storage position	In-out	Position in *Buf[]* in which to store *In*	Depends on data type.	---	---
Buf[] (array)	Input value storage array	In-out	Array to store *In* values	Depends on data type.	---	---
Q	Calculation completed flag	In-out	TRUE: *BufSize* elements or more have been stored in *Buf[]* FALSE: *BufSize* elements are not yet stored in *Buf[]*	Depends on data type.	---	---
Out	Calculation result	Output	Calculation result	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied.A building error will occur.

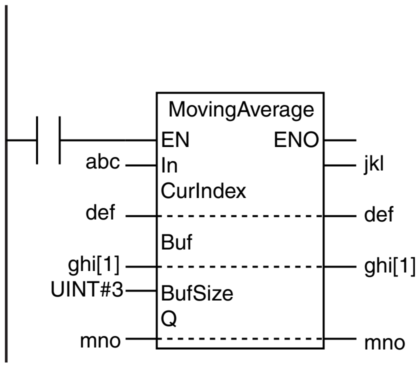
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
BufSize							OK													
CurIndex							OK													
Buf[] (array)	Must be an array with elements that have the same data type as *In*.																			
Q	OK																			
Out						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					

Function

The MovingAverage instruction stores the value of input value *In* in input value storage array *Buf[]* each time it is executed. It stores the average of the stored values in calculation result *Out*. Specify the maximum number of elements to include in the average with *BufSize*.

The processing procedure when *BufSize* is UINT#3 is described below as an example. The instruction and statement are written as follows:

LD



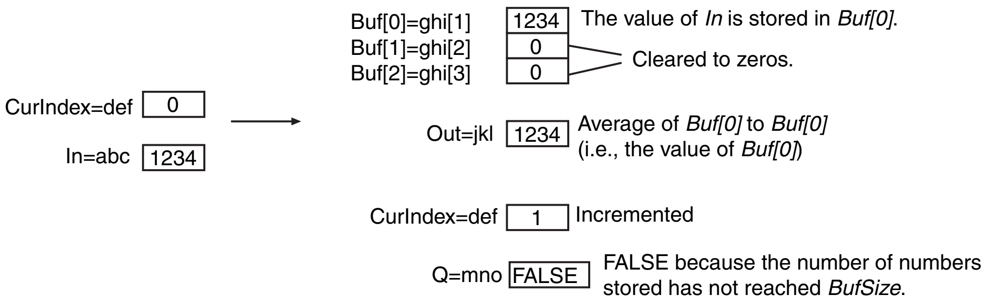
ST

```
jkl:=MovingAverage(abc, def, ghi[1], UINT#3, mno);
```

First Time a Number Is Input

The input value storage position *CurIndex* is set to 0 and the instruction is executed.
Buf[0] to *Buf[BufSize−1]* of input value storage array *Buf[]* are cleared to zeros and the first input value *In* is stored in *Buf[0]*.
The value of calculation completed flag *Q* changes to FALSE. This indicates that the number of values that are stored in *Buf[]* has not reached *BufSize* yet.
While the value of *Q* is FALSE, the average value is calculated for the *CurIndex* + 1 numbers that start from *Buf[0]*. The calculation result is stored in *Out*.
Finally, the value of *CurIndex* is incremented.

First Execution of Instruction

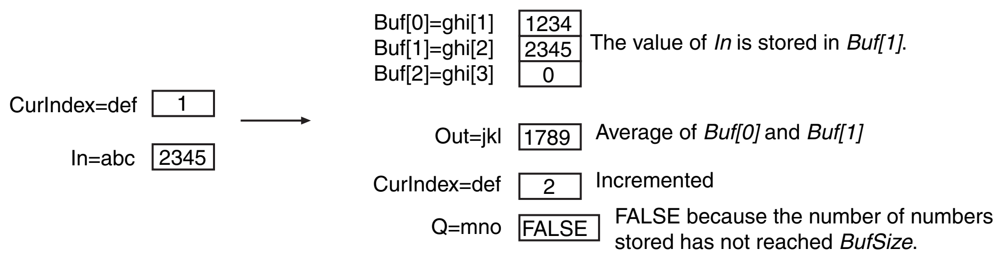


Inputting Numbers Up to *BufSize*

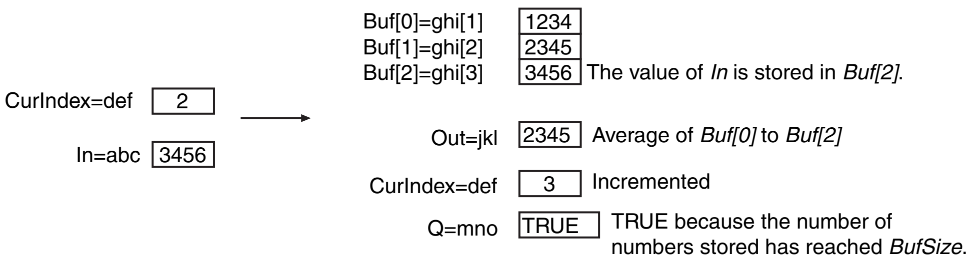
Each time the instruction is executed, the value of *In* is stored in *Buf[CurIndex]*. The average of *CurIndex* + 1 numbers that start from *Buf[0]* is calculated and stored in *Out*.

When the number of instruction executions reaches *BufSize*, the value of *Q* changes to TRUE.

Second Execution of Instruction



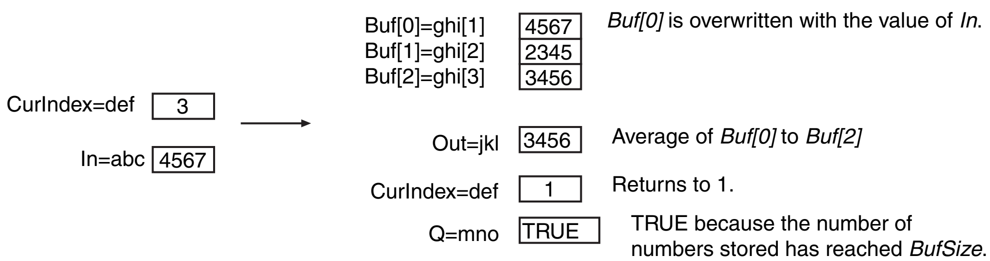
Third Execution of Instruction



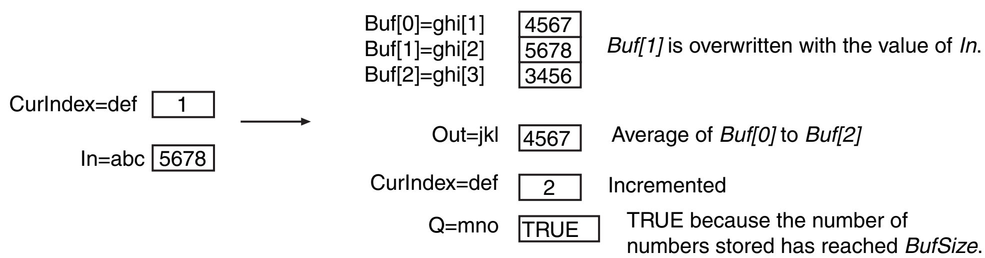
Inputting Numbers after Reaching *BufSize*

Each time the instruction is executed, *Buf[0]* to *Buf[BufSize−1]* are overwritten with the value of *In* in cyclic fashion. The average of *Buf[0]* to *Buf[BufSize−1]* is calculated and stored in *Out*.
The value of *CurIndex* returns to 1 after it reaches *BufSize* and it is then incremented again. The value of *Q* remains TRUE.

Fourth Execution of Instruction



Fifth Execution of Instruction



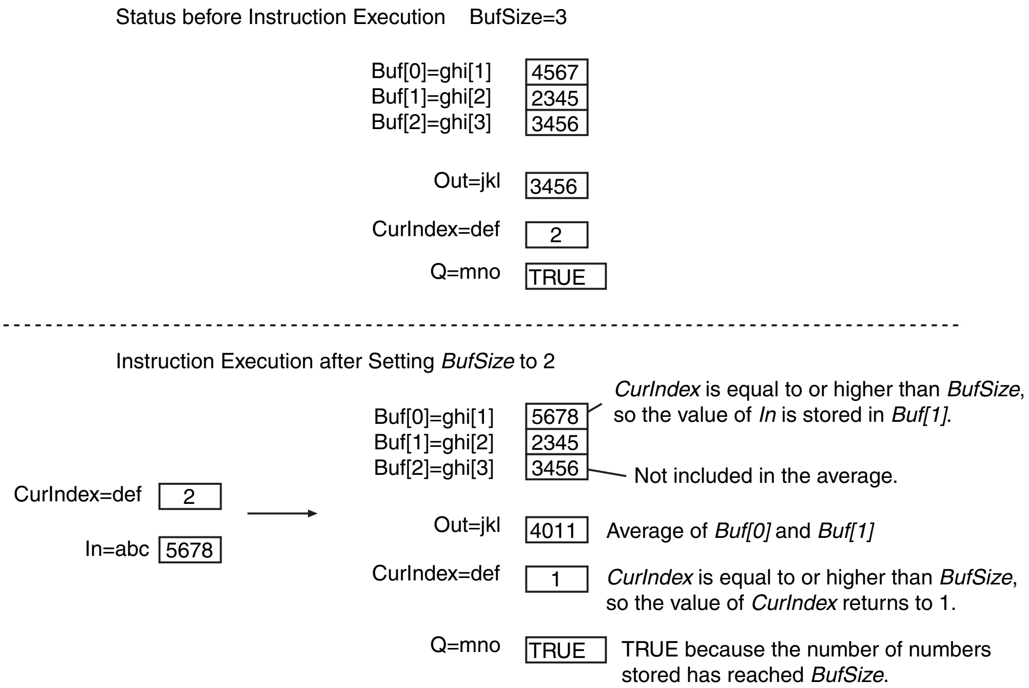
Initializing the Stored Values

If the value of *CurIndex* is set to 0 before the instruction is executed, the values in *Buf[0]* to *Buf[BufSize−1]* are set to 0 and the current value of *In* is stored again in *Buf[0]*.

The value of *CurIndex* changes to 1 and the value of *Q* changes to FALSE.

Changing the Value of *BufSize*

If you change the value of *BufSize* and execute the instruction, operation is performed with the new value of *BufSize* and the current value of *CurIndex*.



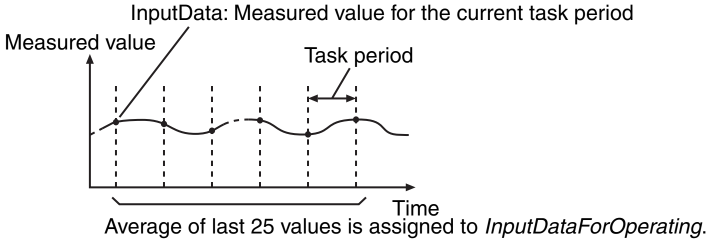
Precautions for Correct Use

- Use the same data type for *In* and the elements of *Buf[]*. If they are different, a building error will occur.
- Use a *Buf[]* array that is at least as large as the value of *BufSize.*
- Even if the calculation result exceeds the valid range of *Out*, an error will not occur. The value of *Out* will be an illegal value.
- If the value of *BufSize* is 0, the values of *Out* and *CurIndex* change to 0. The value of *Q* changes to TRUE.
- If you change the value of *BufSize*, always set the value of *CurIndex* to 0 and initialize the stored values.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of *BufSize* exceeds the size of the *Buf[]* array.

Sample Programming

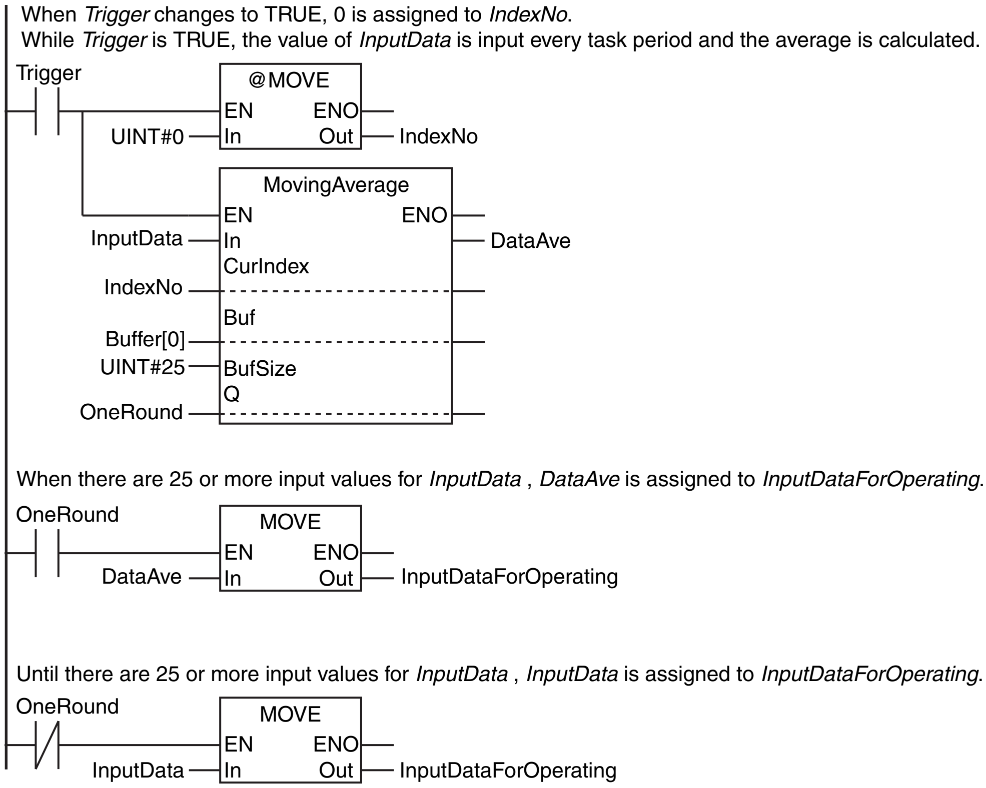
This sample shows how to eliminate the effect of noise and other disturbances in analog input data, e.g., from a sensor. It assigns the average (*DataAve*) of the last 25 values of the input data (*InputData*) to the input data (*InputDataForOperating*) for the next process.
InputData is input every task period as long as the value of the execution condition (*Trigger*) is TRUE. Until 25 values of *InputData* are input, there is not enough data to calculate the average, so the most recent value of *InputData* is assigned to *InputDataForOperating*.

When the value of *Trigger* changes to TRUE, the average is cleared and input of *InputData* is started again from the beginning.



LD

Variable	Data type	Initial value	Comment
Trigger	BOOL	FALSE	Execution condition
InputData	INT	10	Input value
Buffer	ARRAY[0..24] OF INT	[25(0)]	Input value storage array
DataAve	INT	0	Average value
OneRound	BOOL	FALSE	Flag that indicates 25 inputs
IndexNo	UINT	0	Input value storage position
InputDataForOperating	INT	0	Input to next operation



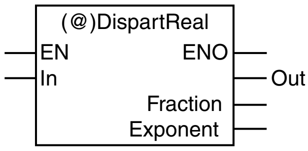
ST

Variable	Data type	Initial value	Comment
Trigger	BOOL	FALSE	Execution condition
LastTrigger	BOOL	FALSE	Value of *Trigger* from previous task period
Operating	BOOL	FALSE	Processing
OperatingStart	BOOL	FALSE	Processing started
Buffer	ARRAY[0..24] OF INT	[25(0)]	Input value storage array
InputData	INT	10	Input value
DataAve	INT	0	Average value
OneRound	BOOL	FALSE	Flag that indicates 25 inputs
IndexNo	UINT	0	Input value storage position
InputDataFor Operating	INT	0	Input to next operation

```
// Detect when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) ) THEN
    OperatingStart:=TRUE;
    Operating:=TRUE;
END_IF;
LastTrigger:=Trigger;

// Clear the average.
IF (OperatingStart=TRUE) THEN
    IndexNo:=UINT#0;
    OperatingStart:=FALSE;
END_IF;

// Calculate the moving average.
IF (Operating=TRUE) THEN
    DataAve:=MovingAverage(
        In      :=InputData,
        CurIndex:=IndexNo,
        Buf     :=Buffer[0],
        BufSize :=UINT#25,
        Q       :=OneRound);

    IF (OneRound=TRUE) THEN
        // Assign the average of last 25 values to InputDataForOperating.
        InputDataForOperating:=DataAve;
    ELSE
        // Assign the most recent value to InputDataForOperating.
        InputDataForOperating:=InputData;
    END_IF;
END_IF;

// End average processing.
IF (Trigger=FALSE) THEN
    Operating:=FALSE;
END_IF;
```

DispartReal

The DispartReal instruction separates a real number into the signed mantissa and the exponent.

Instruction	Name	FB/FUN	Graphic expression	ST expression
DispartReal	Separate Mantissa and Exponent	FUN		Out:=DispartReal(In, Fraction, Exponent);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Real number	Input	Real number to separate	Depends on data type.	---	*1
Out	Return value	Output	Always TRUE	TRUE only	---	---
Fraction	Signed mantissa		Signed mantissa	*2		
Exponent	Exponent		Exponent	*3		

*1 If you omit the input parameter, the default value is not applied. A building error will occur.

*2 The valid ranges depend on the data types of *In* and *Fraction*. Refer to *Function* for details.

*3 If *In* is REAL data, –44 to 32. If *In* is LREAL data, –322 to 294

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														OK	OK					
Out	OK																			
Fraction	Must be DINT if the data type of *In* is REAL and LINT if the data type of *In* is LREAL.																			
Exponent											OK									

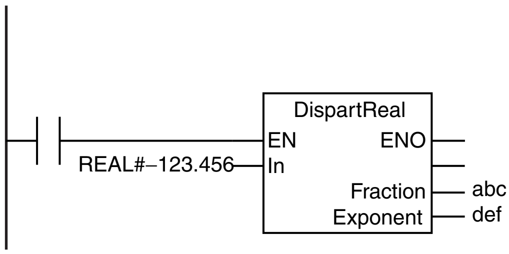
Function

The DispartReal instruction separates real number *In* into signed mantissa *Fraction* and exponent *Exponent*.

If *In* is REAL data, *Fraction* is a 7-digit integer. If *In* is LREAL data, *Fraction* is a 15-digit integer.

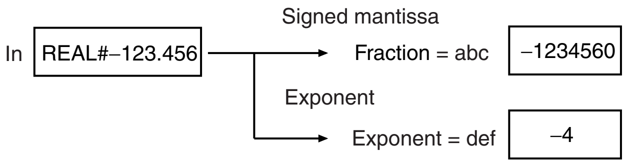
The following example is for when *In* is REAL data with a value of REAL#–123.456.

LD



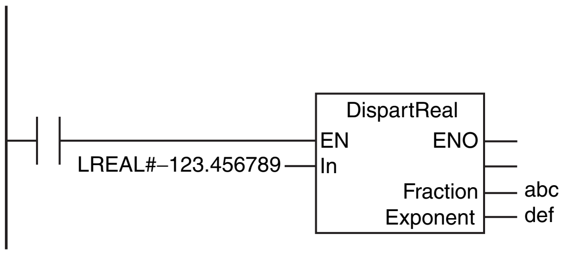
ST

DispartReal(REAL#–123.456, abc, def);



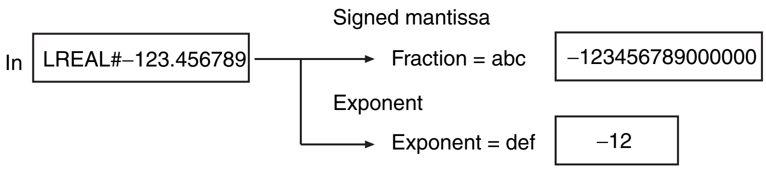
The following example is for when *In* is LREAL data with a value of LREAL#–123.456789.

LD



ST

DispartReal(LREAL#–123.456789, abc, def);



The following table shows the valid ranges for *Fraction* according to the data types *In* and *Fraction*.

Data type of *In*	Data type of *Fraction*	Valid range of *Fraction*
REAL	DINT	–9999999 to 9999999
LREAL	LINT	–999999999999999 to 999999999999999

Additional Information

Use the UniteReal instruction (page 2-444) to combine a signed mantissa and exponent to form a real number.

Precautions for Correct Use

- Depending on the value of *In*, error may occur in the conversion to an integer.

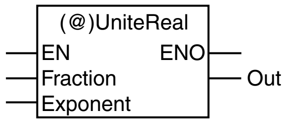
- If the number of valid digits in *In* exceeds the number of valid digits of *Fraction*, the value is rounded to fit in the valid range of *Fraction*. The following table shows how values are rounded.

Value of fractional part	Treatment	Examples
Less than 0.5	The fractional part is truncated.	1.49 → 1 –1.49 → –1
0.5	If the ones digit is an even number, the value is truncated. If it is an odd number, the value is rounded up.	1.50 → 2 2.50 → 2 –1.50 → –2 –2.50 → –2
Greater than 0.5	The fractional part is rounded up.	1.51 → 2 –1.51 → –2

- An error occurs in the following case. *ENO* will be FALSE, and *Fraction* and *Exponent* will not change.
  - The value of *In* is nonnumeric or infinity.

UniteReal

The UniteReal instruction combines a signed mantissa and exponent to make a real number.

Instruction	Name	FB/FUN	Graphic expression	ST expression
UniteReal	Combine Real Number Mantissa and Exponent	FUN		Out:=UniteReal(Fraction, Exponent);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Fraction	Signed mantissa	Input	Signed mantissa	Depends on data type.	---	*
Exponent	Exponent		Exponent			0
Out	Real number	Output	Real number	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

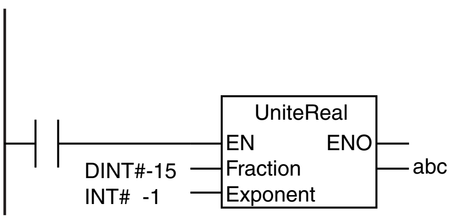
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Fraction												OK	OK							
Exponent											OK									
Out	Must be REAL if the data type of *Fraction* is DINT and LREAL if the data type of *Fraction* is LINT.																			

Function

The UniteReal instruction combines signed mantissa *Fraction* and exponent *Exponent* to make real number *Out*.

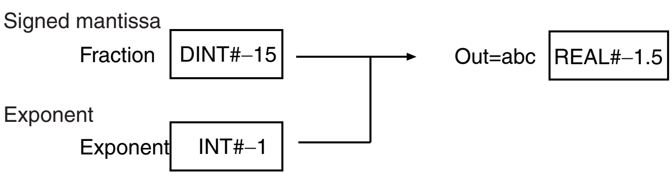
The following example is for when *Fraction* is DINT#–15 and *Exponent* is INT#–1.

LD



ST

abc:=UniteReal(DINT#15, INT#-1);



Additional Information

Use the DispartReal instruction (page 2-441) to separate a real number into the signed mantissa and exponent.

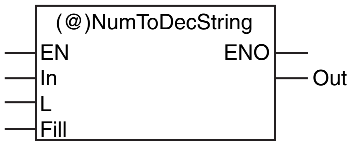
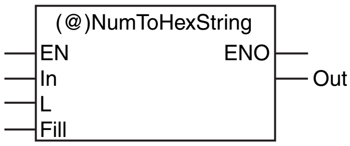
Precautions for Correct Use

- Depending on the values of *Fraction* and *Exponent*, error may occur in the conversion from an integer to a real number.
- If the combined result exceeds the valid range of *Out* and *Exponent* is positive, the value of *Out* will be infinity with the same sign as *Fraction*. If *Exponent* is negative, the value of *Out* will be 0.

NumToDecString and NumToHexString

NumToDecString: Converts an integer to a fixed-length decimal text string.

NumToHexString: Converts an integer to a fixed-length hexadecimal text string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
NumToDecString	Fixed-length Decimal Text String Conversion	FUN		Out:=NumToDecString(In, L, Fill);
NumToHexString	Fixed-length Hexadecimal Text String Conversion	FUN		Out:=NumToHexString(In,L, Fill);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Integer	Input	Integer	Depends on data type.	---	*
L	Number of characters		Number of characters in *Out*	0 to 1985		1
Fill	Fill character		Fill character	_BLANK or _ZERO		_BLANK
Out	Text string	Output	Text string	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						OK	OK	OK	OK	OK	OK	OK	OK							
L							OK													
Fill	Refer to *Function* for the enumerators for the enumerated type _eFILL_CHR.																			
Out																				OK

Function

● NumToDecString

The NumToDecString instruction converts integer *In* to a decimal text string of UTF-8 alphanumeric characters. If *In* contains a negative value, a minus sign (–) is added to the front of the text string.

● NumToHexString

The NumToHexString instruction converts integer *In* to a hexadecimal text string of UTF-8 alphanumeric characters. If *In* is negative, it is expressed in its two's complement (bits inverted and then 1 added).

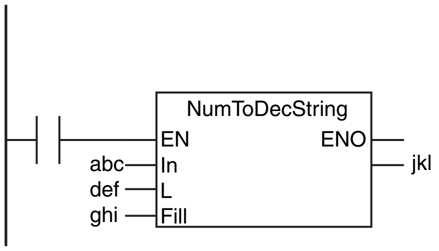
For either instruction, the number of characters in text string *Out* is adjusted to number of characters *L*. If there are not enough characters, the upper digits are filled with fill character *Fill*. If the number of characters in the conversion result exceeds *L*, *L* characters from the lower digits of the conversion result are assigned to *Out*. The NULL character is not included in the number of characters.

The data type of *Fill* is enumerated type _eFILL_CHR. The meaning of the enumerators are as follows:

Enumerator	Meaning
_BLANK	'' (blank character)
_ZERO	'0'

The following examples are for the NumToDecString instruction.

LD



ST

```
jkl:=NumToDecString(abc, def, ghi);
```

In = abc = INT#128, L = def = UINT#8, Fill = ghi = _BLANK

Out = jkl `     128`

In = abc = INT#–128, L = def = UINT#8, Fill = ghi = _BLANK

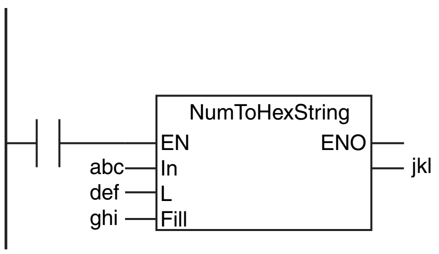
Out = jkl `    –128`

In = abc = INT#–128, L = def = UINT#8, Fill = ghi = _ZERO

Out = jkl `–0000128`

The following examples are for the NumToHexString instruction.

LD



ST

```
jkl:=NumToHexString(abc, def, ghi);
```

In = abc = INT#128, L = def = UINT#8, Fill = ghi = _BLANK

Out = jkl `      80`

In = abc = INT#128, L = def = UINT#8, Fill = ghi = _ZERO

Out = jkl `00000080`

In = abc = INT#–128, L = def = UINT#8, Fill = ghi = _BLANK

Out = jkl `FFFFFF80`

Precautions for Correct Use

- If the value of *L* is 0, *Out* is a text string containing only the NULL character.
- If the number of characters in the conversion result exceeds the value of *L*, *L* characters from the lower characters of the conversion result are stored in *Out*. The following is an example.

Instruction	Value of *In*	Value of *L*	Value of *Out*
NumToDecString	128	2	28
NumToHexString			80

- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of *L* is outside of the valid range.
  - The value of *Fill* is outside of the valid range.

HexStringToNum_**

The HexStringToNum_** instruction converts a hexadecimal text string to an integer.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Hex-StringToNum_**	Hexadecimal Text String-to-Number Conversion Group	FUN	(@)HexStringToNum_** EN ENO In Out "**" must be an integer data type.	Out:=HexStringToNum_**(In); "**" must be an integer data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Hexadecimal text string	Input	Hexadecimal text string	Depends on data type.	---	''
Out	Integer	Output	Integer	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
Out						OK	OK	OK	OK	OK	OK	OK	OK							

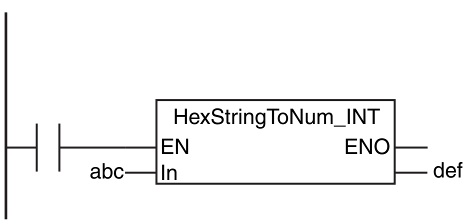
Function

The HexStringToNum_** instruction converts hexadecimal text string *In* to an integer. Any spaces (16#20) or '0' (16#30) in the upper digits are ignored. Underbars (16#5F) in the text string are ignored.

The name of the instruction is determined by the data type of *Out*. For example, if *Out* is the INT data type, the instruction is HexStringToNum_INT.

A few examples are given below.

LD



ST

def:=HexStringToNum_INT(abc);

In = abc | | | | | | |8|0| → Out = def = 128

In = abc | | | | | |–|8|0| → Out = def = –128

In = abc |–|0|0|0|0|0|0|F| → Out = def = –15

Precautions for Correct Use

- Even if the conversion result exceeds the valid range of *Out*, an error will not occur. The value of *Out* will be an illegal value.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The content of *In* includes characters that cannot be converted to numbers.

FixNumToString

The FixNumToString instruction converts a signed fixed-decimal number to a decimal text string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FixNumToString	Fixed-decimal Number-to-Text String Conversion	FUN	(@)FixNumToString EN — ENO In — Out Zero	Out:=FixNumToString(In, Zero);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Fixed-decimal number	Input	Signed fixed-decimal number	Depends on data type.	---	0
Zero	Zero augmenta-tion		Augmentation of zeros if there are less than 3 decimal digits TRUE: Add ‘0’ FALSE: Do not add ‘0’			TRUE
Out	Decimal text string	Output	Decimal text string	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In				OK																
Zero	OK																			
Out																				OK

Function

The FixNumToString instruction converts signed fixed-decimal number *In* to a decimal text string. The following conversion is used.

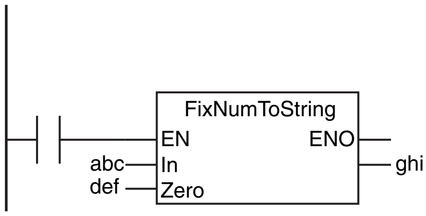
1 The hexadecimal number *In* is converted to a decimal number.

2 The result is divided by 1,000.

Zero augmentation *Zero* specifies whether to add ‘0’ to the third decimal place of *Out* when there are less than three decimal digits in *In*. If the value of *Zero* is TRUE, ‘0’ is added. A NULL character is placed at the end of *Out.*

A few examples are given below.

LD



ST

ghi:=FixNumToString(abc, def);

In **= abc**	***Out*** **= ghi**	
	Zero **= def = TRUE**	***Zero*** **= def = FALSE**
16#0001462C (10#83500)	‘83.500’	‘83.5’
16#00051AA4 (10#334500)	‘334.500’	‘334.5’
16#0003BEFC (10#245500)	‘245.500’	‘245.5’

Additional Information

The format for fixed-point decimal numbers is the same as the fixed-decimal output format of the OMRON FZ-series Vision Sensors.

StringToFixNum

The StringToFixNum instruction converts a decimal text string to a signed fixed-decimal number.

Instruction	Name	FB/FUN	Graphic expression	ST expression
StringToFixNum	Text String-to-Fixed-decimal Conversion	FUN	(@)StringToFixNum EN ENO In Out	Out:=StringToFixNum(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Decimal text string	Input	Decimal text string	Depends on data type.	---	''
Out	Fixed-decimal number	Output	Fixed-decimal number	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
Out				OK																

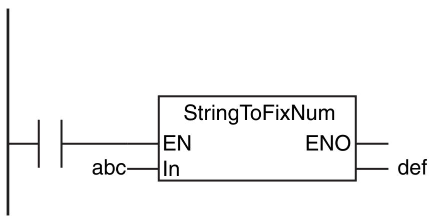
Function

The StringToFixNum instruction converts decimal text string *In* to a fixed-decimal number. The following conversion is used.

1. The number in *In* is multiplied by 1,000.
2. The fractional part is truncated.
3. The result is given as a 32-bit hexadecimal number (DWORD).

A few examples are given below.

LD

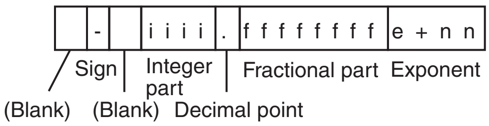


ST

```
def:=StringToFixNum(abc);
```

In = abc	Out = def
‘83.5’	16#0001462C (10#83500)
‘334.5’	16#00051AA4 (10#334500)
‘245.5’	16#0003BEFC (10#245500)

The format of the text sting in *In* is given below.



Name	Format
Sign	• Any consecutive blank characters (16#20) at the beginning of the text string are ignored. Any single plus or minus sign that follows is treated as the sign. • The sign can be omitted. • Any consecutive blank characters after the sign are ignored.
Integer part	• Consecutive numbers (‘0’ to ‘9’) after the sign and up to the decimal point are taken as the integer part. The sign may sometimes be omitted. There may be blank characters between the sign and the integer part. • If the decimal point and fractional part are omitted, the characters up to the exponent are taken as the integer part. • If the decimal point, fractional part, and exponent are omitted, the characters up to the end of the text string are taken as the integer part. • The integer part cannot be omitted. • The maximum number of digits in the integer part is the maximum text string length of 1986 minus the total number of bytes in the following: the sign, decimal point, fractional part, exponent, and blank characters before and after the sign.
Decimal point	• A single dot (‘.’) following the integer part is taken as the decimal point. • Omit the decimal point if there is no fractional part.
Fractional part	• Consecutive numbers (‘0’ to ‘9’) after the decimal point and up to the exponent are taken as the fractional part. • If the exponent is omitted, the characters up to the end of the text string are taken as the fractional part. • The fractional part can be omitted. If there is no decimal point, then there is no fractional part. • The fractional part can consist of a maximum of 15 digits.
Exponent	• The exponent consists of a single ‘e’ or ‘E’ after the fractional part, a following single plus or minus sign, and the remaining continuous numbers (‘0’ to ‘9’) to the end of the text string. • If there is no fractional part, then the above text string after the decimal point is taken as the exponent. • If there is no decimal point or fractional part, then the above text string after the integer part is taken as the exponent. • The exponent can be omitted. • The numeric part of the exponent can consist of a maximum of three digits.

Example 1: The following example uses the sign, decimal point, and fractional part, but does not use an exponent.

Example 2: The following example uses the sign, decimal point, fractional part, and exponent.

In '| + | 1 | . | 2 3 4 5 6 7 | e + 0 2 |' ⟶ Out | 0001E240 |

Example 3: The following example does not use the sign, but uses the decimal point, fractional part, and exponent.

In '| | 1 2 3 4 5 | . | 6 7 | e - 0 2 |' ⟶ Out | 0001E240 |

Example 4: The following example does not use the sign, fractional part, decimal point, and exponent.

In '| 1 |' ⟶ Out | 00003E8 |

Additional Information

The format for fixed-point decimal numbers is the same as the fixed-decimal output format of the OMRON FZ-series Vision Sensors.

Precautions for Correct Use

- The digits after the third decimal digit are truncated in *In*.
- Underbars (16#5F) in the text string in *In* are ignored.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The content of *In* includes characters that cannot be converted to numbers.
  - The content of *In* has a decimal point but not a fractional part.

DtToString

The DtToString instruction converts a date and time to a text string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
DtToString	Date and Time-to-Text String Conversion	FUN	(@)DtToString EN ENO In Out	Out:=DtToString(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Date and time	Input	Date and time	Depends on data type.	Year, month, day, hour, minutes, seconds	DT#1970-1-1-0:0:0
Out	Text string	Output	Text string	30 bytes (29 single-byte alphanumeric characters plus the final NULL character)	---	---

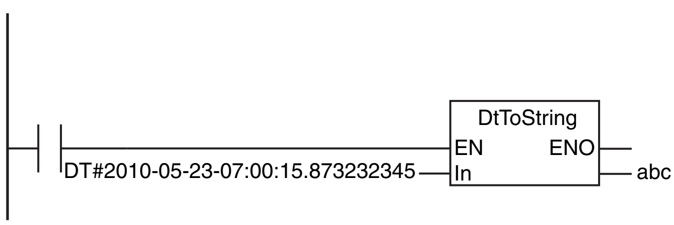
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																			OK	
Out																				OK

Function

The DtToString instruction converts date and time *In* to a text string. A NULL character is placed at the end of text string *Out.*

An example when *In* is 2010-5-23-07:00:15.873232345 (7:00 am and 15.873232345 seconds on May 23, 2010) is given below. The value of variable *abc* will be ‘2010-05-23-07:00:15.873232345’.

LD



ST

abc:=DtToString(DT#2010-05-23-07:00:15.873232345);

The DtToString instruction converts date and time *In* to a text string.
The value of *In* is 7:00 am and 15.873232345 seconds on May 23, 2010, so the value of *abc* will be ‘2010-05-23-07:00:15.873232345’.

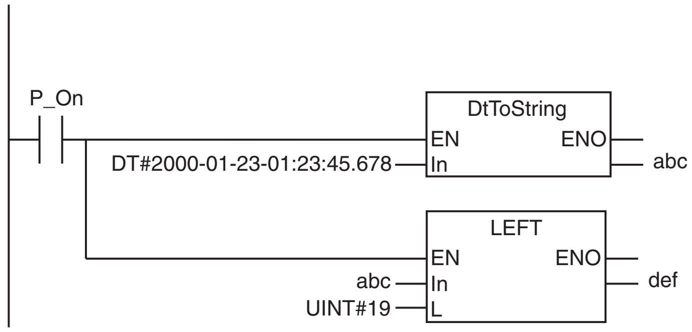
In DT#2010-05-23-07:00:15.873232345 → Converted to text string. → Out=abc '2010-05-23-07:00:15.873232345'

Additional Information

Out is in nanoseconds. To get a text string in seconds or milliseconds, combine this instruction with the LEFT or RIGHT instruction (page 2-556).

An example to get a text string in seconds is given below.

- LD

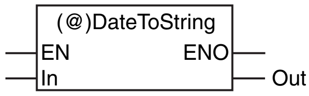


- ST

```
def:=LEFT(DtToString(DT#2000-01-23-01:23:45.678), UINT#19);
```

DateToString

The DateToString instruction converts a date to a text string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
DateToString	Date-to-Text String Conversion	FUN		Out:=DateToString(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Date	Input	Date	Depends on data type.	Year, month, day	D#1970-1-1
Out	Text string	Output	Text string	11 bytes (10 single-byte alphanumeric characters plus the final NULL character)	---	---

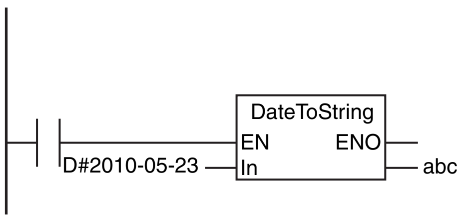
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																	OK			
Out																				OK

Function

The DateToString instruction converts date *In* to a text string. A NULL character is placed at the end of *Out.*

An example when *In* is 2010-5-23 (May 23, 2010) is given below. The value of variable *abc* will be ‘2010-05-23’.

LD



ST

abc:=DateToString(D#2010-05-23);

The DateToString instruction converts date *In* to a text string.
The value of *In* is May 23, 2010, so the value of *abc* will be ‘2010-05-23’.

In D#2010-05-23 —Converted to text string.→ Out=abc '2010-05-23'

TodToString

The TodToString instruction converts a time of day to a text string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TodToString	Time of Day-to-Text String Conversion	FUN	(@)TodToString EN ENO In Out	Out:=TodToString(In);

Variables

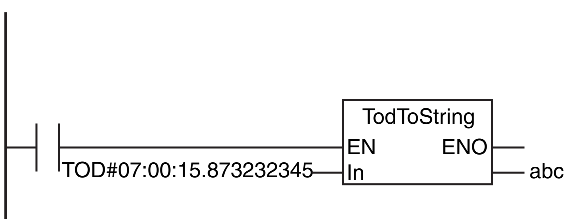
Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Time of day	Input	Time of day	Depends on data type.	Hour, minutes, seconds	TOD#0:0:0
Out	Text string	Output	Text string	19 bytes (18 single-byte alpha-numeric characters plus the final NULL character)	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																		OK		
Out																				OK

Function

The TodToString instruction converts time of day *In* to a text string. A NULL character is placed at the end of *Out.* An example when *In* is 07:00:15.873232345 (7:00 am and 15.873232345 seconds) is given below. The value of variable *abc* will be ‘07:00:15.873232345’.

LD



ST

abc:=TodToString(TOD#07:00:15.873232345);

The TodToString instruction converts time of day *In* to a text string.
The value of *In* is 7:00 am and 15.873232345 seconds, so the value of *abc* will be ‘07:00:15.873232345’.

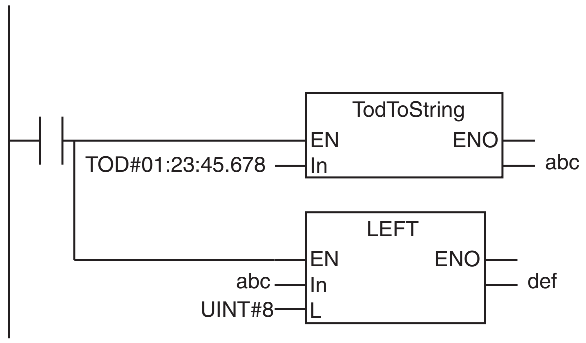
In [TOD#07:00:15.873232345] —Converted to text string.→ Out=abc ['07:00:15.873232345']

Additional Information

Out is in nanoseconds. To get a text string in seconds or milliseconds, combine this instruction with the LEFT or RIGHT instruction (page 2-556).

An example to get a text string in seconds is given below.

- LD



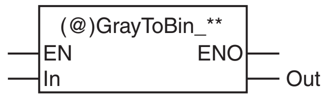
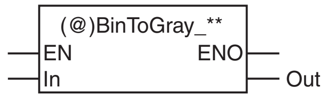
- ST

```
def:=LEFT(TodToString(TOD#01:23:45.678), UINT#8);
```

GrayToBin_** and BinToGray_**

GrayToBin_**: Converts a gray code to a bit string.

BinToGray_**: Converts a bit string to a gray code.

Instruction	Name	FB/FUN	Graphic expression	ST expression
GrayToBin_**	Gray Code-to-Binary Code Conversion Group	FUN	"**" must be a bit string data type.	Out:=GrayToBin_**(In); "**" must be a bit string data type.
BinToGray_**	Binary Code-to-Gray Code Conversion	FUN	"**" must be a bit string data type.	Out:=BinToGray_**(In); "**" must be a bit string data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	Depends on data type.	---	0
Out	Conversion result	Output	Conversion result	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Out	Must be same data type as *In*																			

Function

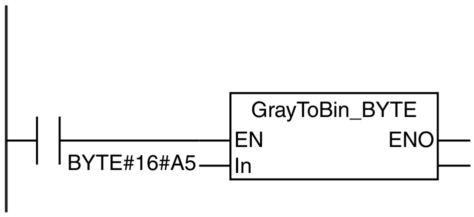
● GrayToBin_**

The GrayToBin_** instructions convert the gray code in date to convert *In* to a bit string. The conversion procedure is as follows for when *In* and *Out* are BYTE data.

1. The most-significant bit (bit 7) of *In* is used as is as the most-significant bit (bit 7) of *Out.*
2. An exclusive logical OR is taken of the value of bit 6 in *In* and the value of bit 7 in *Out*. The result is used as bit 6 of *Out*.
3. This process is repeated through the least-significant bit (bit 0) of *Out*.

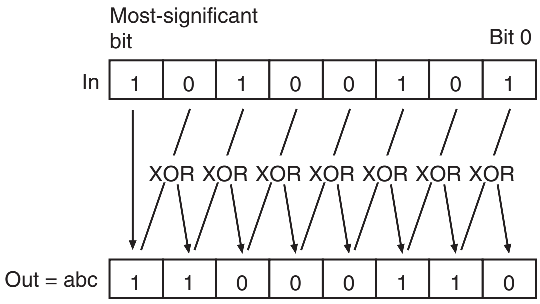
The following example for the GrayToBin_BYTE instruction is for when *In* is BYTE#16#A5.

LD



ST

```
abc:=GrayToBin_BYTE(BYTE#16#A5);
```



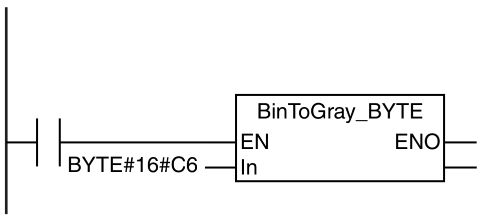
BinToGray_**

The BinToGray_** instructions convert the bit string in data to convert *In* to a gray code. The conversion procedure is as follows for when *In* and *Out* are BYTE data.

1. The most-significant bit (bit 7) of *In* is used as is as the most-significant bit (bit 7) of *Out.*
2. An exclusive logical OR is taken of the value of bit 7 in *In* and the value of bit 6 in *In*. The result is used as bit 6 of *Out*.
3. This process is repeated through the least-significant bit (bit 0) of *Out*.

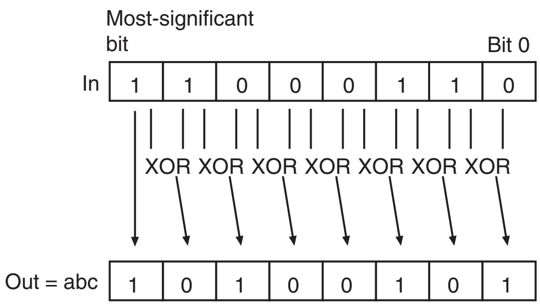
The following example for the BinToGray_BYTE instruction is for when *In* is BYTE#16#C6.

LD



ST

```
abc:=BinToGray_BYTE(BYTE#16#C6);
```



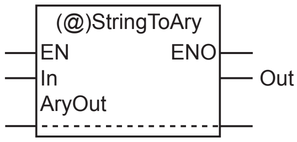
The name of the instruction is determined by the data types of *In* and *Out*. For example, if *In* and *Out* are the WORD data type, the instruction is GrayToBin_WORD or BinToGray_WORD.

Precautions for Correct Use

The data types of *In* and *Out* must be the same.

StringToAry

The StringToAry instruction converts a text string to a BYTE array.

Instruction	Name	FB/FUN	Graphic expression	ST expression
StringToAry	Text String-to-Array Conversion	FUN		Out:=StringToAry(In, AryOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Text string	Input	Text string	Depends on data type.	---	''
AryOut[] (array)	BYTE array	In-out	BYTE array	Depends on data type.	---	---
Out	Number of bytes to convert	Output	Number of bytes to convert	0 to 1985	Bytes	---

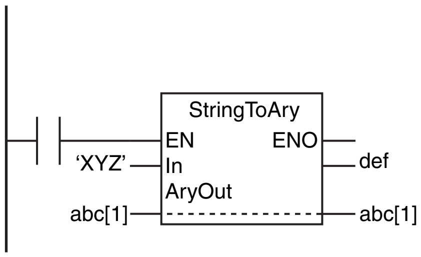
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
AryOut[] (array)		OK																		
Out							OK													

Function

The StringToAry instruction takes the character codes in text string *In* as numbers and stores them individually in a BYTE array, *AryOut[]*. The number of bytes that was converted is stored in *Out*.

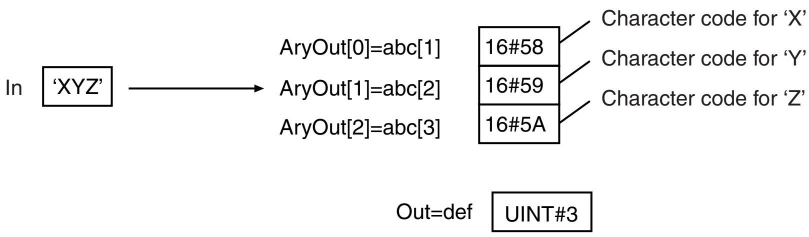
The following example is for when *In* is ‘XYZ’.

LD



ST

```
def:=StringToAry('XYZ', abc[1]);
```



Precautions for Correct Use

- The NULL character at the end of *In* is not stored in *AryOut[]*.
- If the *In* text string contains only the NULL character, the value of *Out* will be 0 and *AryOut[]* will not change.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* and *AryOut[]* will not change.
  - The number of bytes in *In* is larger than the number of elements in *AryOut[]*.

AryToString

The AryToString instruction converts a BYTE array to a text string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryToString	Array-to-Text String Conversion	FUN	(@)AryToString EN ENO In Out Size	Out:=AryToString(In, Size);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	BYTE array	Input	BYTE array Maximum number of elements: 1985	Depends on data type.	---	*
Size	Number of elements to convert	Input	Number of elements of *In[]* for conversion	0 to 1985	---	1
Out	Text string	Output	Text string	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

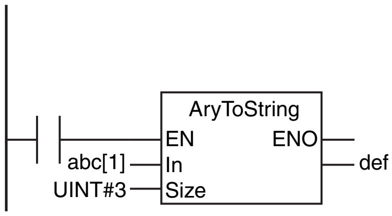
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)		OK																		
Size							OK													
Out																				OK

Function

The AryToString instruction takes the elements of a BYTE array, *In[]*, from *In[0]* as character codes and stores them in text string *Out*. A NULL character is placed at the end of *Out*. *Size* specifies the number of elements of *In[]* to convert. If there is a NULL character between *In[0]* and *In[Size*–1*]*, no character codes past it are stored in *Out*.

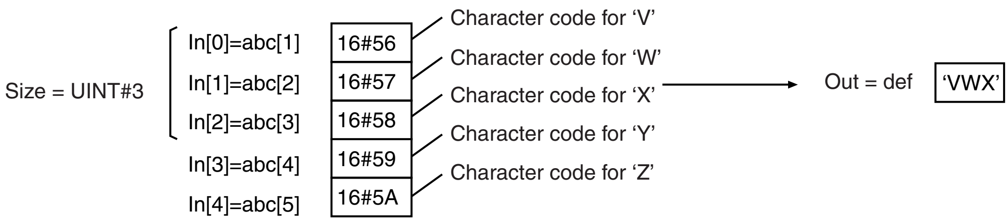
The following example is for when *Size* is UINT#3.

LD



ST

```
def:=AryToString(abc[1], UINT#3);
```



Precautions for Correct Use

- If the value of *Size* is 0, *Out* is a text string containing only the NULL character.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of *Size* exceeds the array area of *In[]*.

DispartDigit

The DispartDigit instruction separates a bit string into 4-bit units.

Instruction	Name	FB/FUN	Graphic expression	ST expression
DispartDigit	Four-bit Separation	FUN	(@)DispartDigit EN ENO In Out Num AryOut	DispartDigit(In, Num, AryOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to separate	Input	Bit string to separate	Depends on data type.	---	*
Num	Number of digits to separate		Number of digits to separate	0 to No. of bits in *In*		1
AryOut[] (array)	Separation results array	In-out	Separation results array	16#00 to 16#0F	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Num						OK														
AryOut[] (array)		OK																		
Out	OK																			

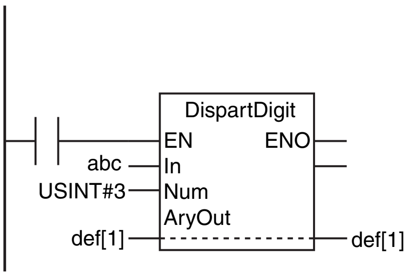
Function

The DispartDigit instruction separates data to separate *In* into 4-bit units (digits) and stores them in separation results array *AryOut[]*.

First, *In* is separated into 4-bit units. Then, the lowest 4 bits are stored in *AryOut[0]*. *AryOut[0]* is BYTE data, so 16#0 is stored in bits 4 to 7. This process is repeated for the number of digits that is specified in number of digits to separate *Num*.

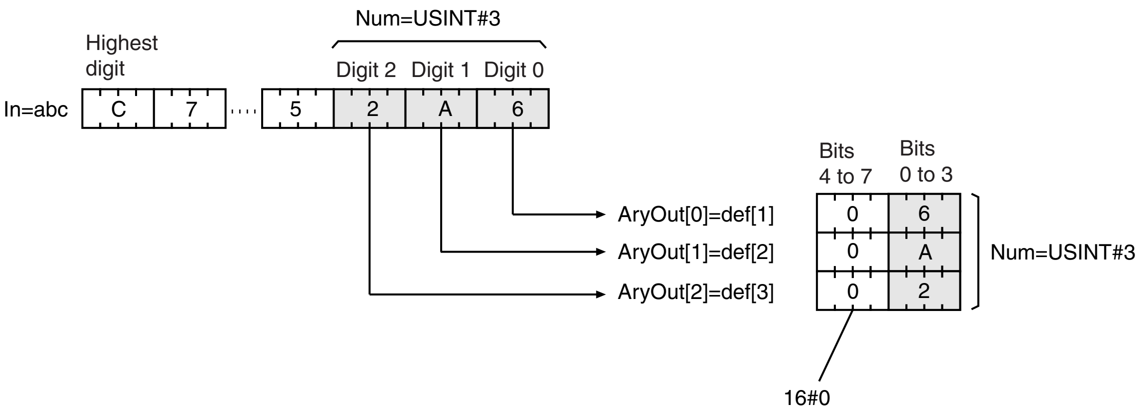
The following example is for when *Num* is USINT#3.

LD



ST

```
DispartDigit(abc, USINT#3, def[1]);
```



Additional Information

Use the UniteDigit_** instruction (page 2-469) to join 4-bit units from array elements.

Precautions for Correct Use

- The values in *AryOut[]* do not change if the value of *Num* is 0.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *AryOut[]* will not change.
  - The value of *Num* is outside of the valid range.
  - The value of *Num* exceeds the array area of *AryOut[]*.

UniteDigit_**

The UniteDigit_** instructions join 4-bit units of data into a bit string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
UniteDigit_**	Four-bit Join Group	FUN	(@)UniteDigit_** EN ENO In Out Num "**" must be a bit string data type.	Out:=UniteDigit_**(In, Num); "**" must be a bit string data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array to join	Input	Array to join	Depends on data type.	---	*
Num	Number of digits to join		Number of digits to join	0 to No. of bits in *Out*		1
Out	Joined result	Output	Bit string with joined result	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)		OK																		
Num						OK														
Out		OK	OK	OK	OK															

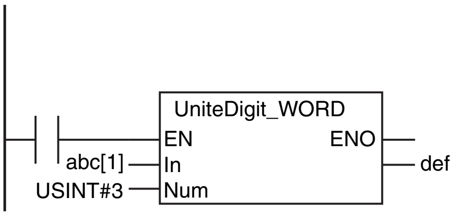
Function

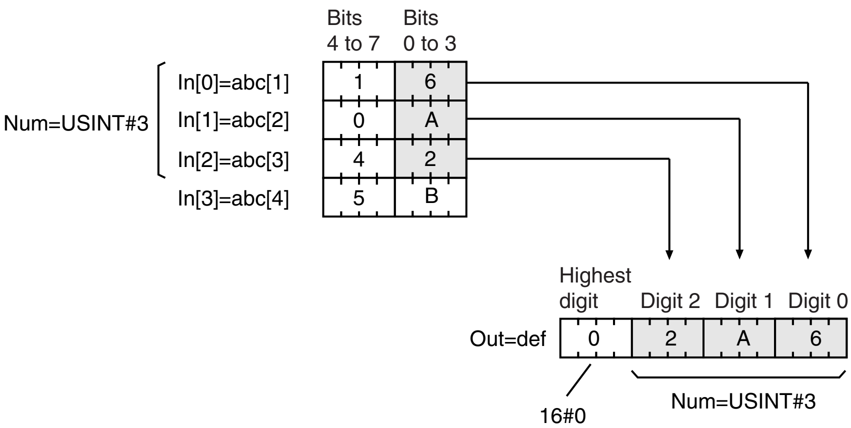
The UniteDigit_** instructions join 4-bit units from the elements of array to join *In[]*. It creates a bit string in joined result *Out*. (Four bits is one digit.)

Number of digits to join *Num* specifies the number of array elements to join. First, the lower four bits from each element from *In[0]* to *In[Num*–1*]* are joined to create a bit string with *Num* digits. To this, 16#0 is added to the upper digits for the number of digits of *Out* minus the value of *Num*. The result is stored in *Out*.

The name of the instruction is determined by the data type of *Out*. For example, if *Out* is the WORD data type, the instruction is UniteDigit_WORD.

The following example shows the UniteDigit_WORD instruction when *Num* is USINT#3.

LD	ST
	def:=UniteDigit_WORD(abc[1], USINT#3);



Additional Information

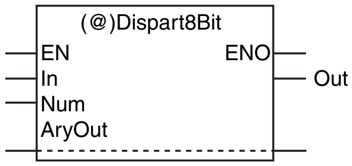
Use the DispartDigit instruction (page 2-467) to separate a bit string into 4-bit units.

Precautions for Correct Use

- If the value of Num is 0, the value of *Out* is 0.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of *Num* is outside of the valid range.
  - The value of *Num* exceeds the array area of *In[]*.

Dispart8Bit

The Dispart8Bit instruction separates a bit string into individual bytes.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Dispart8Bit	Byte Data Separation	FUN		Dispart8Bit(In, Num, AryOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to separate	Input	Bit string to separate	Depends on data type.	---	*
Num	Number of bytes to separate		Number of bytes to separate	0 to No. of bytes in *In*		1
AryOut[] (array)	Separation results array	In-out	Separation results array	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Num						OK														
AryOut[] (array)		OK																		
Out	OK																			

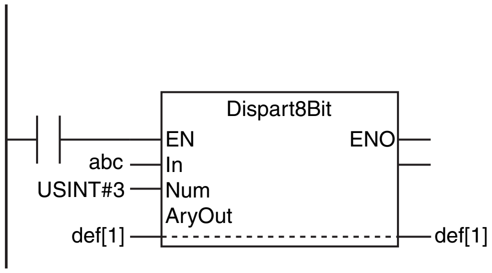
Function

The Dispart8Bit instruction separates data to separate *In* into individual bytes and stores them in separation results array *AryOut[]*.

First, *In* is separated into bytes. Then, the lowest byte is stored in *AryOut[0].* Then, the next byte is stored in *AryOut[1].* This process is repeated for the number of bytes that is specified in number of bytes to separate *Num*.

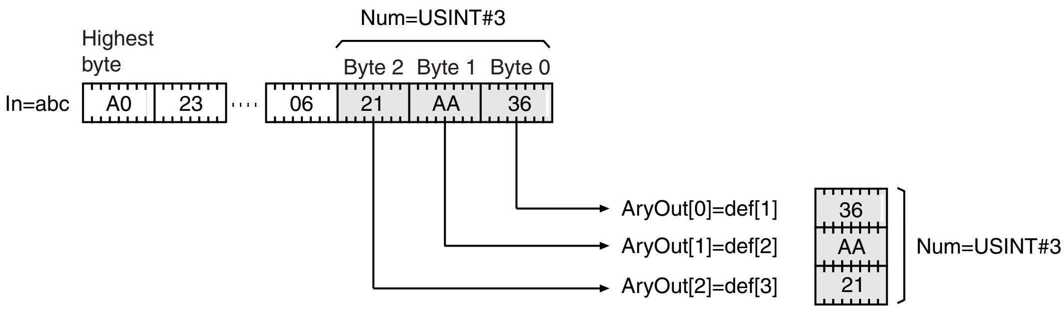
The following example is for when *Num* is USINT#3.

LD



ST

Dispart8Bit(abc, USINT#3, def[1]);



Additional Information

Use the Unite8Bit_** instruction (page 2-473) to join 1-byte units from array elements.

Precautions for Correct Use

- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *AryOut[]* will not change.
  - The value of *Num* is outside of the valid range.
  - The value of *Num* exceeds the number of bytes in *In*.

Unite8Bit_**

The Unite8Bit_** instructions join bytes of data into a bit string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Unite8Bit_**	Byte Data Join Group	FUN	 "**" must be a bit string data type.	Out:=Unite8Bit_**(In, Num); "**" must be a bit string data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array to join	Input	Array to join	Depends on data type.	---	*
Num	Number of bytes to join		Number of bytes to join	0 to No. of bytes in *Out*		1
Out	Joined result	Output	Bit string with joined result	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)		OK																		
Num						OK														
Out		OK	OK	OK	OK															

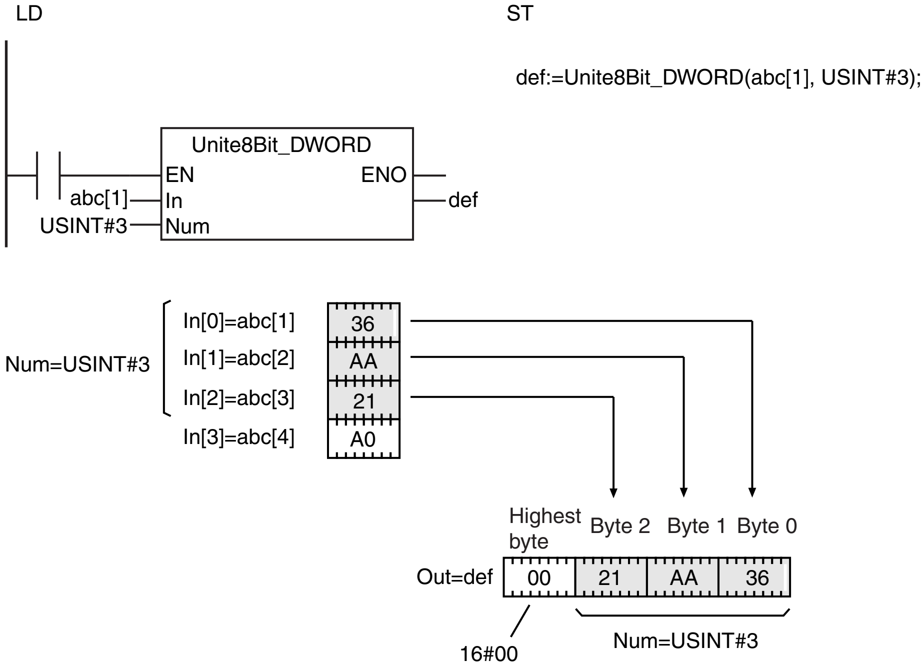
Function

The Unite8Bit_** instructions join elements of array to join *In[]* to create a bit string in joined result *Out*.

Number of bytes to join *Num* specifies the number of array elements to join. First, *In[0]* to *In[Num*–1*]* are joined to create a bit string with *Num* bytes. To this, 16#00 is added to the upper bytes for the number of bytes of *Out* minus the value of *Num*. The result is stored in *Out*.

The name of the instruction is determined by the data type of *Out*. For example, if Out is the DWORD data type, the instruction is Unite8Bit_DWORD.

The following example shows the Unite8Bit_DWORD instruction when *Num* is USINT#3.



Additional Information

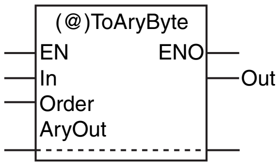
Use the Dispart8Bit instruction (page 2-471) to separate a bit string into 1-byte units.

Precautions for Correct Use

- If the value of Num is 0, the value of *Out* is 0.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of *Num* is outside of the valid range.
  - The value of *Num* exceeds the array area of *In[]*.

ToAryByte

The ToAryByte instruction separates a variable into bytes and stores the bytes in a BYTE array.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ToAryByte	Conversion to Byte Array	FUN		Out:=ToAryByte(In, Order, AryOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Data to convert	Depends on data type.	---	*
Order	Conversion order		Conversion order	_LOW_HIGH or _HIGH_LOW		_LOW _HIGH
AryOut[] (array)	Conversion results array	In-out	Conversion results array	Depends on data type.	---	---
Out	Number of elements in result	Output	Number of elements in result	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	An enumeration, array, array element, structure, or structure member can also be specified.																			
Order	Refer to *Function* for the enumerators for the enumerated type _eBYTE_ORDER.																			
AryOut[] (array)		OK																		
Out							OK													

Function

The ToAryByte instruction separates the value of data to convert *In* into individual bytes and stores them in order in conversion results array *AryOut[]* starting from *AryOut[0].* Number of elements in result *Out* contains the number of elements stored in *AryOut[].*

Conversion order *Order* specifies the order in which to convert the value of *In* to bytes. The data type of *Order* is enumerated type _eBYTE_ORDER. The meaning of the enumerators are as follows:

Enumerator	Meaning
_LOW_HIGH	Lower byte first, higher byte last
_HIGH_LOW	Higher byte first, lower byte last

When the Data Type of *In* Is Two Bytes or Larger

If the data type of *In* is two bytes or larger, *In* is separated into bytes and stored in *AryOut[]*. The following data types have two bytes or more.

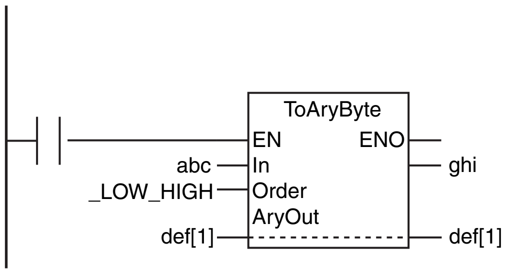
Classification	Data type
Bit strings	WORD, DWORD, and LWORD
Integers	UINT, UDINT, ULINT, INT, DINT, and LINT
Real numbers	REAL and LREAL
Times, durations, dates, and text strings	TIME, DATE, TOD, DT, and STRING types of two bytes or more
Others	An enumeration, an array for which the total for all elements is 2 bytes or more, an array element that is 2 bytes or more, a structure for which the total for all members is 2 bytes or more, or a structure member that is 2 bytes or more

The processing procedure is as follows:

1. First, the value in *In* is separated into words (two bytes).
2. The lowest word is separated into bytes.
3. If *Order* is *_LOW_HIGH*, the lower byte is stored in *AryOut[0]* and the higher byte is stored in *AryOut[1].* If *Order* is *_HIGH_LOW*, the higher byte is stored in *AryOut[0]* and the lower byte is stored in *AryOut[1].*
4. The next word is separated into bytes and stored in *AryOut[2]* and *AryOut[3]* in the same way.
5. This process is repeated to the end of the value of *In*. If *In* is an array, the same process is repeated to the last element in *In*.

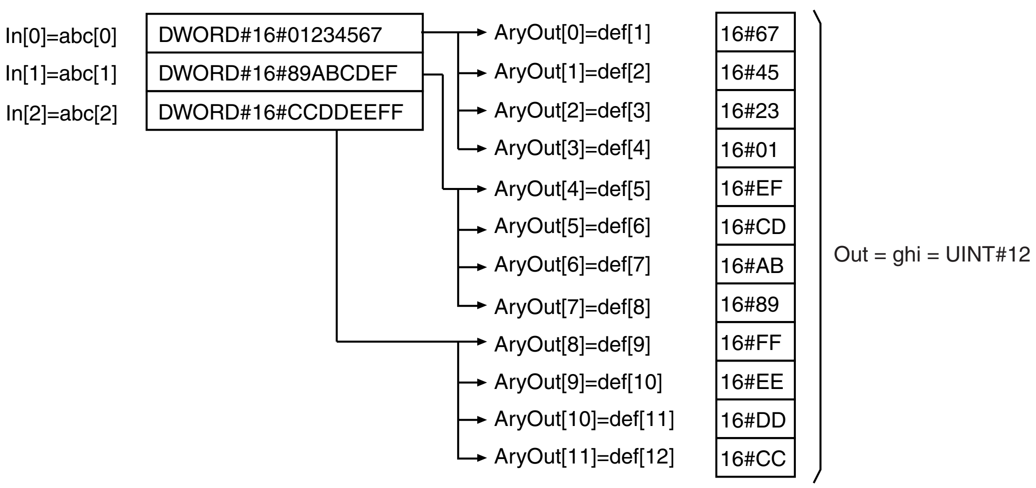
The following example is for when *In* is a DWORD array with three elements and *Order* is *_LOW_HIGH*.

LD

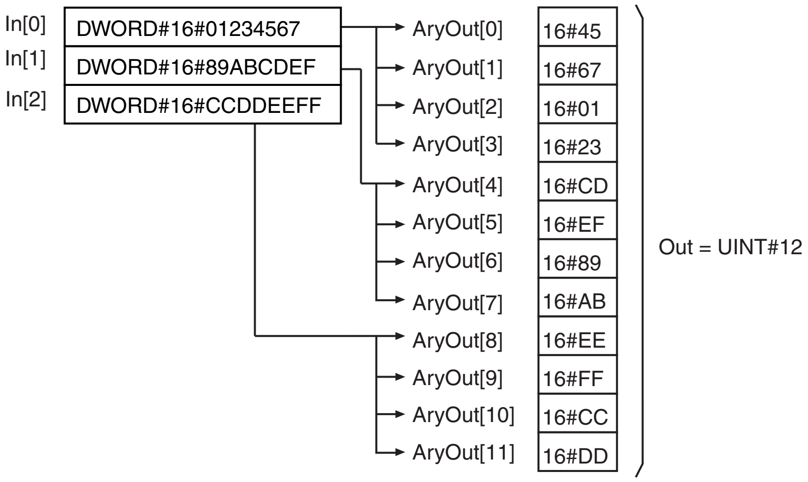


ST

```
ghi:=ToAryByte(abc, _LOW_HIGH, def[1]);
```



The following example is for when *In* is the same as above and *Order* is *_HIGH_LOW*.



When the Data Type of *In* Is One Byte

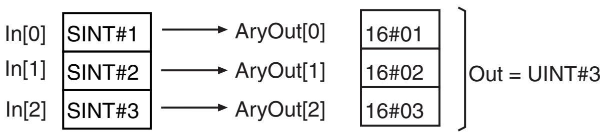
If the data type of *In* is one byte, *In* is stored in *AryOut[]* as one byte. The following data types have one byte.

Classification	Data type
Bit strings	BYTE
Integers	USINT and SINT
Real numbers	None
Times, durations, dates, and text strings	STRING types with one byte
Others	An array for which the total for all elements is 1 byte, an array element that is 1 byte, a structure for which the total for all members is 1 byte, or a structure member that is 1 byte.

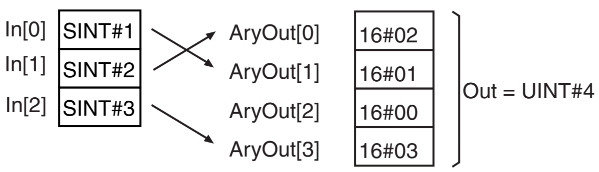
The following storage method is used.

Value of *Order*	*In* (array or not)	Storage method in *AryOut[]*
_LOW_HIGH	Not an array	Value of *In* is stored in *AryOut[0].*
	Array	Value of *In[i]* is stored in *AryOut[i].*
_HIGH_LOW	Not an array	Value of *In* is stored in *AryOut[1].* 16#00 is stored in *AryOut[0].*
	Array	*In[i]* (where *i* is even) is stored in *AryOut[i+1].* *In[i]* (where *i* is odd) is stored in *AryOut[i–1].* If the number of elements in *In[]* is odd, 16#00 is stored last in *AryOut[n–1].*

The following example is for when *In* is a SINT array with three elements and *Order* is *_LOW_HIGH*.



The following example is for when *In* is the same as above and *Order* is *_HIGH_LOW*.

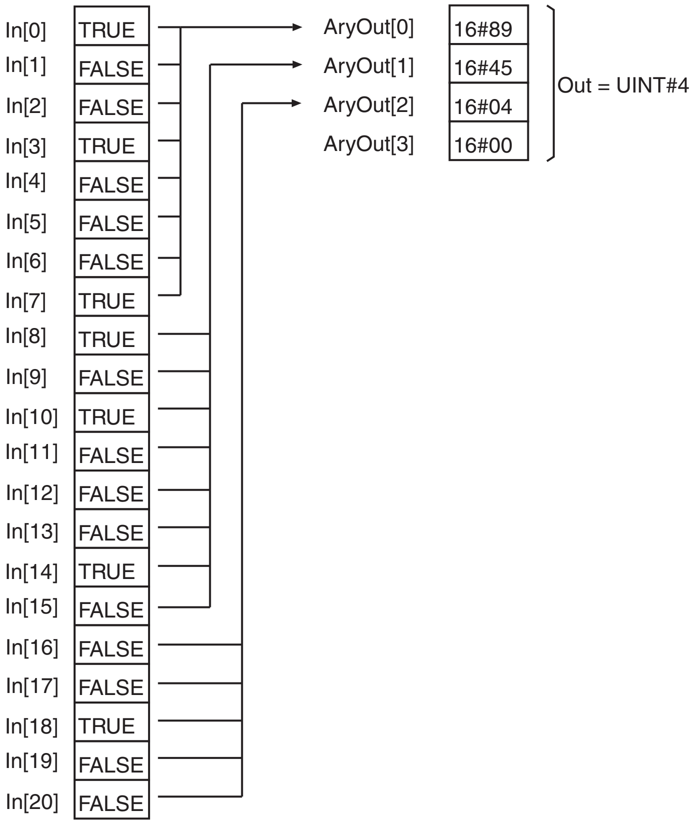


When *In* Is BOOL Data

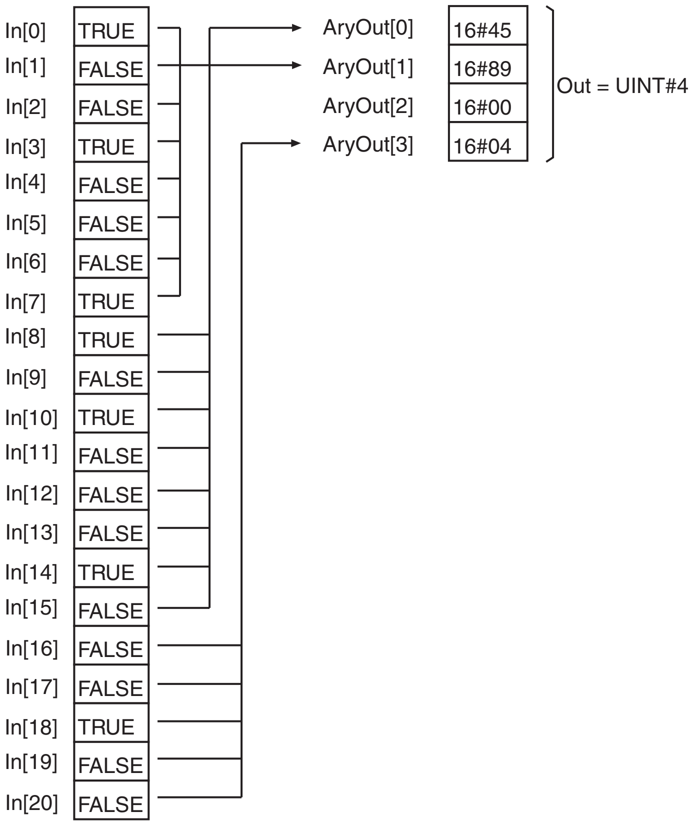
If the data type of *In* is BOOL (one bit), data is stored in *AryOut[]* as described below.

Value of *Order*	*In* (array or not)	Storage method in *AryOut[]*
_LOW_HIGH	Not an array	The logical OR of the value of *In* and 16#00 is stored in *AryOut[0].*
	Array	Values of *In[0]* to *In[7]* are joined and stored in *AryOut[0].* Values of *In[8]* to *In[15]* are joined and stored in *AryOut[1].* The same process is repeated to store the rest of the data. If there is not sufficient data in *In[]* for 8 values, FALSE is added to the most-significant bit. The value of *Out* is always even. If there are not sufficient bit values, the remaining values will all be FALSE.
_HIGH_LOW	Not an array	The logical OR of the value of *In* and 16#00 is stored in *AryOut[1].* 16#00 is stored in *AryOut[0]*
	Array	Values of *In[0]* to *In[7]* are joined and stored in *AryOut[1].* Values of *In[8]* to *In[15]* are joined and stored in *AryOut[0].* The same process is repeated to store the rest of the data. The value of *Out* is always even. If there are not sufficient bit values, the remaining values will all be FALSE.

The following example is for when *In* is a BOOL array with 21 elements and *Order* is *_LOW_HIGH*.



The following example is for when *In* is the same as above and *Order* is *_HIGH_LOW*.

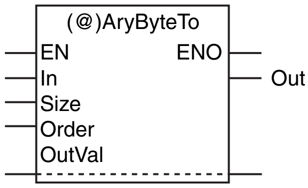


Precautions for Correct Use

- Always use a variable for the input parameter to pass to *In*. A building error will occur if a constant is passed.
- If *In* is an enumeration, you cannot directly pass an enumerator to it. A building error will occur if an enumerator is passed to it directly.
- If *In* is STRING data, the text string is not converted to numbers. The contents of the variable is taken as a bit string and converted to a byte array.
- If *In* is a structure, adjustment areas between members may be inserted into AryOut[].
- If the value of *Order* is *_HIGH_LOW* and the total number of bytes in *In* is an odd number, 16#00 is added to the end of *In* to make an even number of bytes before the conversion is started.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* and *AryOut[]* will not change.
  - The value of *Order* is outside of the valid range.
  - The conversion result exceeds the array area of *AryOut[]*.

AryByteTo

The AryByteTo instruction joins BYTE array elements and stores the result in a variable.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryByteTo	Conversion from Byte Array	FUN		AryByteTo(In, Size, Order, OutVal);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array to convert		Array to convert			*
Size	Number of elements to convert	Input	Number of elements in *In[]* to convert	Depends on data type.	---	1
Order	Conversion order		Conversion order	_LOW_HIGH or _HIGH_LOW		_LOW _HIGH
OutVal	Conversion result	In-out	Conversion result	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)		OK																		
Size							OK													
Order	Refer to *Function* for the enumerators for the enumerated type _eBYTE_ORDER.																			
OutVal	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	An enumeration, array, array element, structure, or structure member can also be specified.																			
Out	OK																			

Function

The AryByteTo instruction takes the first *Size* elements in array to convert *In[]* and joins them to match the size of the data type of conversion result *OutVal*. It then stores the result in *OutVal*.

Order specifies the order to join the elements of *In[]*. The data type of *Order* is enumerated type _eBYTE_ORDER. The meaning of the enumerators are as follows:

Enumerators	Meaning
_LOW_HIGH	Lower byte first, higher byte last
_HIGH_LOW	Higher byte first, lower byte last

When the Data Type of *OutVal* Is Two Bytes or Larger

If the data type of *OutVal* is two bytes or larger, elements from *In[]* are joined so that the result is just large enough for the size of the data type of *OutVal*. The result is stored in *OutVal*. The following data types have two bytes or more.

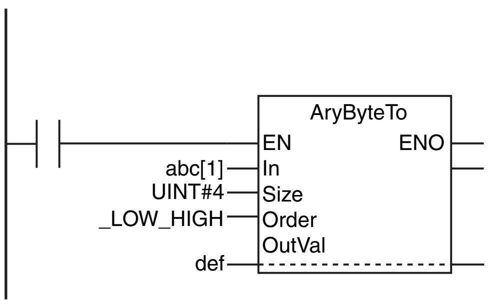
Classification	Data type
Bit strings	WORD, DWORD, and LWORD
Integers	UINT, UDINT, ULINT, INT, DINT, and LINT
Real numbers	REAL and LREAL
Times, durations, dates, and text strings	TIME, DATE, TOD, DT, and STRING types of two bytes or more
Others	An enumeration, an array for which the total for all elements is 2 bytes or more, an array element that is 2 bytes or more, a structure for which the total for all members is 2 bytes or more, or a structure member that is 2 bytes or more

The processing procedure is as follows:

1. *In[0]* and *In[1]* are joined according to the value of *Order* to create one word (two bytes) of data. If *Order* is *_LOW_HIGH*, the higher byte is stored in *In[1]* and the lower byte is stored in *In[0].* If *Order* is *_HIGH_LOW*, the higher byte is stored in *In[0]* and the lower byte is stored in *In[1].*
2. In the same way elements that start from *In[2]* and *In[3]* are joined to make more words of data.
3. The words of data are joined to match the size of the data type of *OutVal*. For example, if *OutVal* is DWORD data, four individual words of data are joined.
4. The resulting data is stored in *OutVal*.

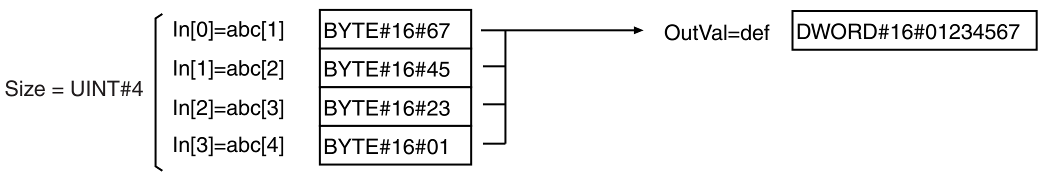
The following example is for when *OutVal* is DWORD data, *Size* is UINT#4, and *Order* is *_LOW_HIGH*.

LD

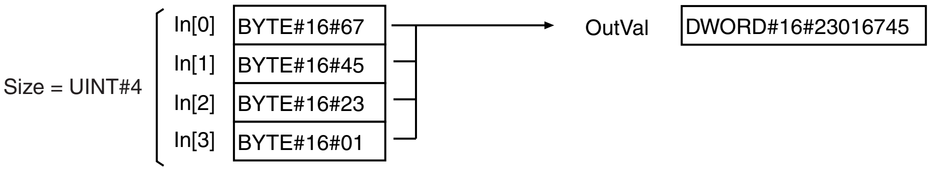


ST

```
AryByteTo(abc[1], UINT#4, _LOW_HIGH, def);
```



The following example is for when *OutVal* is the same as above, *Size* is UINT#4, and *Order* is *_HIGH_LOW*.



When the Data Type of *OutVal* Is One Byte

If the data type of *OutVal* is one byte, one byte of *In[]* is stored directly in *OutVal*.

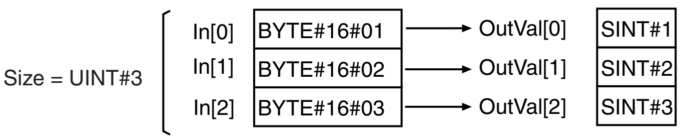
The following data types have one byte.

Classification	Data type
Bit strings	BYTE
Integers	USINT and SINT
Real numbers	None
Times, durations, dates, and text strings	STRING types with one byte
Others	An array for which the total for all elements is 1 byte, an array element that is 1 byte, a structure for which the total for all members is 1 byte, or a structure member that is 1 byte.

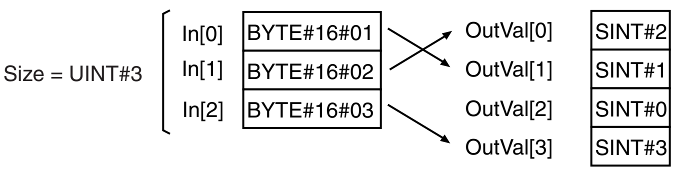
The following storage method is used.

Value of *Order*	*OutVal* (array or not)	Storage method in *OutVal*
_LOW_HIGH	Not an array	Value of *In[0]* is stored in *OutVal*
	Array	Value of *In[i]* is stored in *OutVal[i]*
_HIGH_LOW	Not an array	Value of *In[1]* is stored in *OutVal*
	Array	*In[i]* (where *i* is even) is stored in *OutVal[i+1]*. *In[i]* (where *i* is odd) is stored in *OutVal[i–1]*. If the value of *Size* is odd, data is stored up to *OutVal[Size]* and 16#00 is stored in *OutVal[Size*–1*]*.

The following example is for when *OutVal* is a SINT array with three elements, *Size* is UINT#3, and *Order* is *_LOW_HIGH*.



The following example is for when *OutVal* and *Size* are the same as above and *Order* is *_HIGH_LOW*.

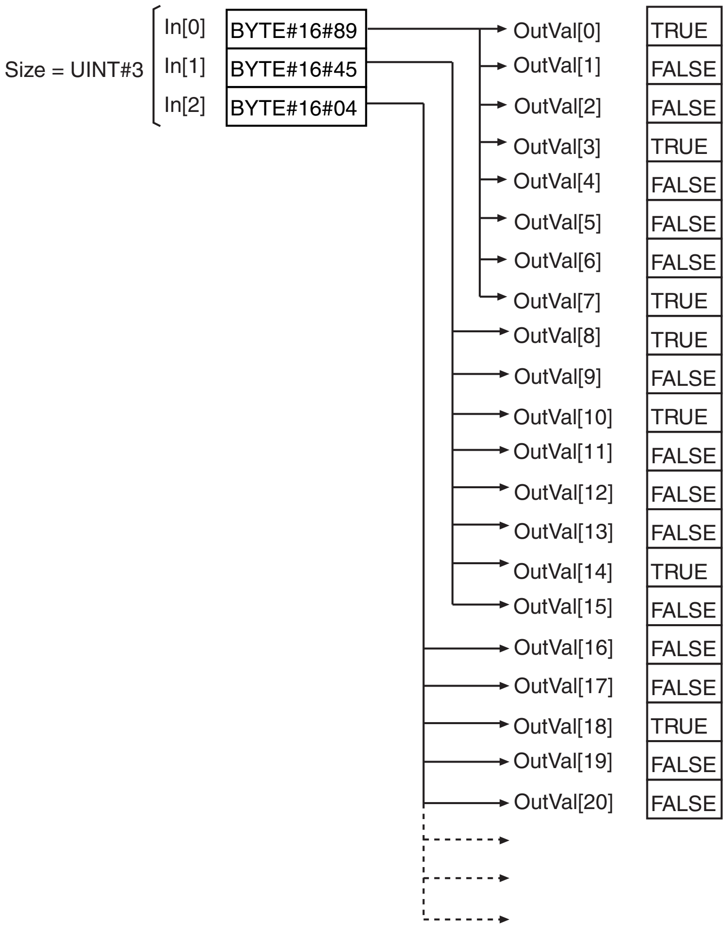


When *OutVal* Is BOOL Data

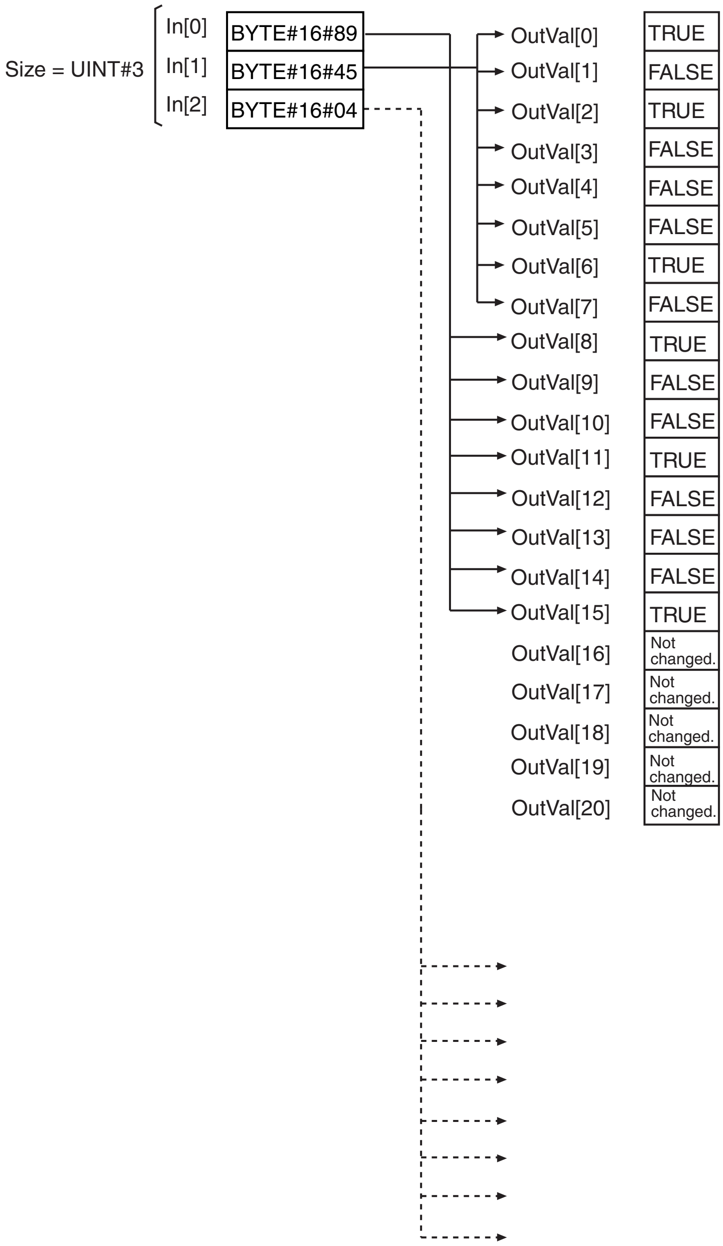
If the data type of *OutVal* is BOOL (one bit), data is stored in *OutVal* as described below.

Value of *Order*	*OutVal* (array or not)	Storage method in *OutVal*
_LOW_HIGH	Not an array	Value of bit 0 of *In[0]* is stored in *OutVal.*
	Array	Value of *In[0]* is separated and stored in *OutVal[0]* to *OutVal[7].* Value of *In[1]* is separated and stored in *OutVal[8]* to *OutVal[15].* The same process is repeated to store the rest of the data. Remaining bits are discarded.
_HIGH_LOW	Not an array	Value of bit 0 of *In[1]* is stored in *OutVal.*
	Array	Value of *In[0]* is separated and stored in *OutVal[8]* to *OutVal[15].* Value of *In[1]* is separated and stored in *OutVal[0]* to *OutVal[7].* The same process is repeated to store the rest of the data. Remaining bits are discarded.

The following example is for when *OutVal* is a BOOL array with 21 elements, *Size* is UINT#3, and *Order* is *_LOW_HIGH*.



The following example is for when *OutVal* and *Size* are the same as above and *Order* is *_HIGH_LOW*.

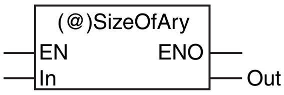


Precautions for Correct Use

- If *OutVal* is a structure, some of the values of *In[]* may be inserted in adjustment areas between members depending on the composition.
- If the value of *Size* is 0, the value of *Out* will be TRUE and *OutVal* will not change.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *OutVal* will not change.
  - The value of *Order* is outside of the valid range.
  - The value of *Size* exceeds the number of elements in *In[]*.

SizeOfAry

The SizeOfAry instruction gets the number of elements in an array.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SizeOfAry	Get Number of Array Elements	FUN		Out:=SizeOfAry(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array	Input	Array	Depends on data type.	---	*
Out	Number of elements	Output	Number of elements	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	Arrays of enumerations or structures can also be specified.																			
Out							OK													

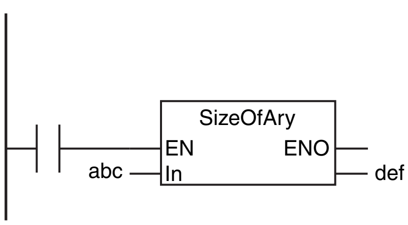
Function

The SizeOfAry instruction gets the number of elements in array *In[]*. For the input parameter, use an array name, such as *array*, and not an array element name, such as *array[0]*.

The following figure shows a programming example.

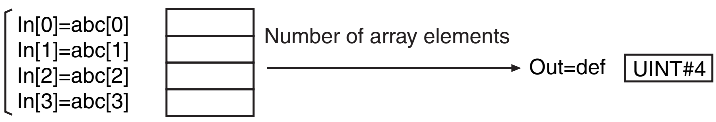
Variable	Data type
abc	ARRAY[0..3] OF INT

LD



ST

def:=SizeOfAry(abc);

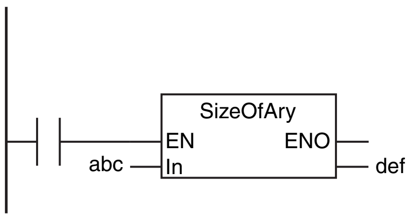


Additional Information

In[] can be an array with two or more dimensions. *Out* will contain the total number of elements for all dimensions of *In[]*. For example, if the input parameter that is passed to *In[]* is *ARRAY[0..1,0..2]*, the value of *Out* will be UINT#6.

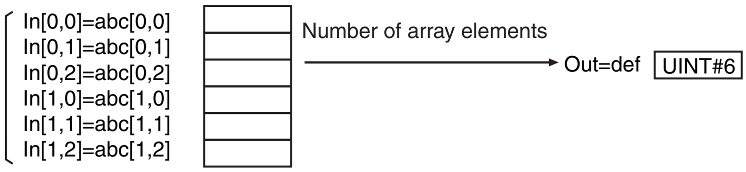
Variable	Data type
abc	ARRAY[0..1,0..2] OF BOOL

LD



ST

```
def:=SizeOfAry(abc);
```



PackWord

The PackWord instruction joins two 1-byte data into a 2-byte data.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
PackWord	2-byte Join	FUN	(@)PackWord EN ENO High Out Low	Out:=PackWord(High,Low);

Version Information

A CPU Unit with unit version 1.12 or later and Sysmac Studio version 1.16 or higher are required to use this instruction.

Variables

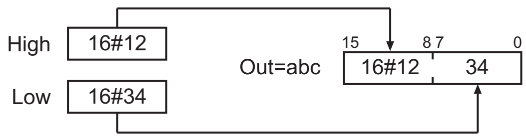
Name	Meaning	I/O	Description	Valid range	Unit	Default
High	Byte data H	Input	Data in bytes stored in bit 15-8	Depends on data type.	---	0
Low	Byte data L		Data in bytes stored in bit 7-0	Depends on data type.	---	0
Out	Joined data	Output	2-byte data	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
High		OK																		
Low		OK																		
Out			OK																	

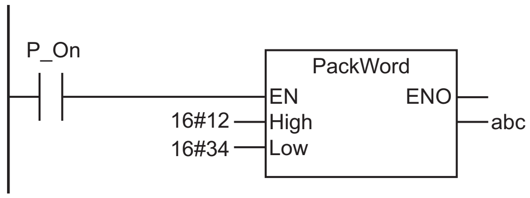
Function

The PackWord instruction joins two 1-byte data into a 2-byte data.
The data specified in *High* is stored in bit 15-8, and the data specified in *Low* is stored in bit 7-0.

The following example shows the instruction when *High* is 16#12 and *Low* is 16#34.
The value of variable abc will be 16#1234.



● LD



● ST

```
abc:=PackWord(16#12,16#34);
```

PackDword

The PackDword instruction joins four 1-byte data into a 4-byte data.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
PackDword	4-byte Join	FUN	(@)PackDword EN ENO HighHigh Out HighLow LowHigh LowLow	Out:=PackDword(HighHigh, HighLow, LowHigh, LowLow);

Version Information

A CPU Unit with unit version 1.12 or later and Sysmac Studio version 1.16 or higher are required to use this instruction.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
HighHigh	Byte data HH	Input	Data in bytes stored in bit 31-24	Depends on data type.	---	0
HighLow	Byte data HL		Data in bytes stored in bit 23-16	Depends on data type.	---	0
LowHigh	Byte data LH		Data in bytes stored in bit 15-8	Depends on data type.	---	0
LowLow	Byte data LL		Data in bytes stored in bit 7-0	Depends on data type.	---	0
Out	Joined data	Output	4-byte data	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
HighHigh		OK																		
HighLow		OK																		
LowHigh		OK																		
LowLow		OK																		
Out				OK																

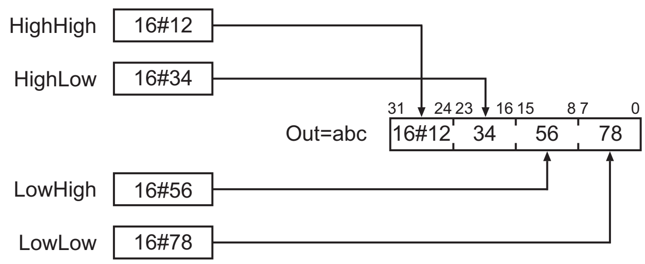
Function

The PackDword instruction joins four 1-byte data into a 4-byte data.

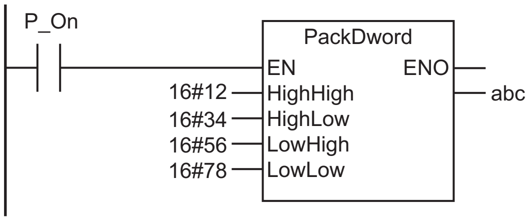
The data specified in *HighHigh* is stored in bit 31-24, the data specified in *HighLow* in bit 23-16, the data specified in *LowHigh* in bit 15-8, and the data specified in *LowLow* in bit 7-0.

The following example shows the instruction when *HighHigh* is 16#12, *HighLow* is 16#34, *LowHigh* is 16#56, and *LowLow* is 16#78.

The value of variable abc will be 16#12345678.



● LD



● ST

abc:=PackDword(16#12,16#34,16#56,16#78);

LOWER_BOUND/UPPER_BOUND

The LOWER_BOUND instruction gets the first number of array dimensions.
The UPPER_BOUND instruction gets the last number of array dimensions.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
LOWER_BOUND	Get First Number of Array	FUN	(@)LOWER_BOUND EN ENO ARR Out DIM	Out:=LOWER_BOUND(ARR, DIM);
UPPER_BOUND	Get Last Number of Array	FUN	(@)UPPER_BOUND EN ENO ARR Out DIM	Out:=UPPER_BOUND(ARR, DIM);

Version Information

A CPU Unit with unit version 1.18 or later and Sysmac Studio version 1.22 or higher are required to use this instruction.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
ARR	Array to process	Input	Specify the array from which to get the first number or last number of array. [a]	---	---	---
DIM	Dimension	Input	Specify the dimension. [b]	---	---	1
Out	Return value	Output	First number or last number	Depends on data type.	---	---

a. Use an array name, such as array, and not an array element name, such as array[0].
b. For one-dimensional array, specify 1.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
ARR	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	Arrays of enumerations or structures can also be specified.																			
DIM												OK								
Out												OK								

Function

The LOWER_BOUND instruction gets the first number of the dimension specified in *DIM* of the array variable specified in *ARR*.

Similarly, the UPPER_BOUND instruction gets the last number of the dimension specified in *DIM* of the array variable specified in *ARR*.

Related System-defined Variables

Name	Meaning	Data type	Description
P_PRGER	Instruction Error Flag	BOOL	TRUE: Error occurred. It remains TRUE until changed to FALSE. FALSE: Set to FALSE by the user program.

Precautions for Correct Use

An error will occur in the following cases. *ENO* will change to FALSE, and *Out* will not change.

- *ARR* is not an array.
- The value specified in *DIM* is 0 or less, or exceeds the range of the dimension of *ARR*.

Sample Programming

Calculating the Sum of an Array

This sample programming shows how to define a one-dimensional variable-length array variable and how to get the first number and last number of the dimension in the variable-length array variable.

● User-defined Function Program (Sum)

Internal Variables	Variable	Data type	Initial value	Comment
	i	DINT		

Input/output variables	Variable	I/O	Data type	Comment
	EN	Input	BOOL	
	ENO	Output	BOOL	
	a	In-out	ARRAY[*] OF INT	

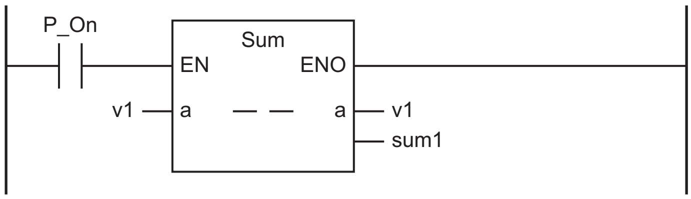
Return value	Variable	Data type	Initial value	Comment
	Sum	INT		

```
Sum := 0;
FOR i := LOWER_BOUND(a,1) TO UPPER_BOUND(a,1) DO
    Sum := Sum + a[i];
END_FOR;
```

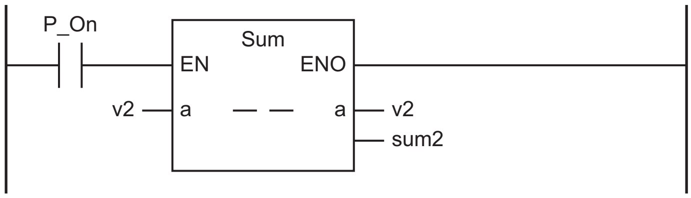
● Calling Program

Internal Variables	Variable	Data type	Initial value	Comment
	v1	ARRAY[0..4] OF INT	[1,2,3,4,5]	
	v2	ARRAY[0..9] OF INT	[1,2,3,4,5,6,7,8,9,10]	
	sum1	INT		
	sum2	INT		

Sum1 = 1+2+3+4+5 =15



Sum2 = 1+2+3+4+5+6+7+8+9+10 =55



Adding 2×2 Matrices

This sample programming shows how to define a multi-dimensional variable-length array variable and how to use the LOWER_BOUND and UPPER_BOUND instructions for the multi-dimensional variable-length array variable.

● User-defined Function Program (Matrix_Add)

Internal Variables	Variable	Data type	Initial value	Comment
	i	DINT		
	j	DINT		
	m1	DINT		
	m2	DINT		
	n1	DINT		
	n2	DINT		

Input/output variables	Variable	I/O	Data type	Comment
	EN	Input	BOOL	
	ENO	Output	BOOL	
	A	In-out	ARRAY[*,*] OF DINT	
	B	In-out	ARRAY[*,*] OF DINT	
	C	In-out	ARRAY[*,*] OF DINT	

Return value	Variable	Data type	Initial value	Comment
	Matrix_Add	BOOL		

```
m1 := LOWER_BOUND(C,1);
m2 := UPPER_BOUND(C,1);
n1 := LOWER_BOUND(C,2);
n2 := UPPER_BOUND(C,2);

FOR i := m1 TO m2 DO
    FOR j := n1 TO n2 DO
        C[i,j] := A[i,j] + B[i,j];
    END_FOR;
END_FOR;
```

● Calling Program

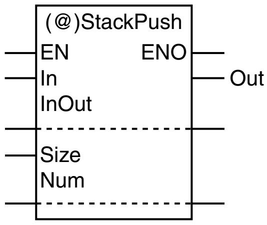
Internal Variables	Variable	Data type	Initial value	Comment
	X	ARRAY[0..1,0..1] OF DINT	[0, 1, 2, 3]	
	Y	ARRAY[0..1,0..1] OF DINT	[1, 2, 3, 4]	
	Z	ARRAY[0..1,0..1] OF DINT		

```
// Z = X + Y = |0 1| + |1 2| = |1 3|
//             |2 3|   |3 4|   |5 7|
Matrix_Add(X, Y, Z);
```

Stack and Table Instructions

StackPush

The StackPush instruction stores a value at the top of a stack.

Instruction	Name	FB/FUN	Graphic expression	ST expression
StackPush	Push onto Stack	FUN		StackPush(In, InOut, Size, Num);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Input value	Input	Value, structure, or structure member to place in the stack	Depends on data type.	---	---
Size	Number of stack elements		Number of stack array elements			1
InOut[] (array)	Stack array	In-out	Array that functions as stack	Depends on data type.	---	---
Num	Number of stored elements		Number of elements stored in stack			
Out	Return value	Output	Always TRUE	TRUE only	---	---

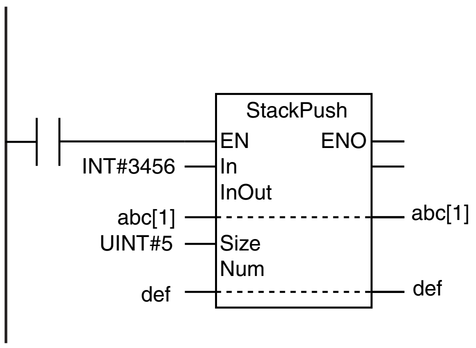
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	An enumeration, structure, or structure member can also be specified.																			
Size							OK													
InOut[] (array)	Must be an array with elements that have the same data type as *In*.																			
Num							OK													
Out	OK																			

Function

The instruction assumes that there are number of stored elements *Num* elements stored in stack array *InOut[]*. Input value *In* is written to the next element, *InOut[Num]*. Then, *Num* is incremented. For number of stack elements *Size*, specify the number of elements in *InOut[]* to use as a stack.

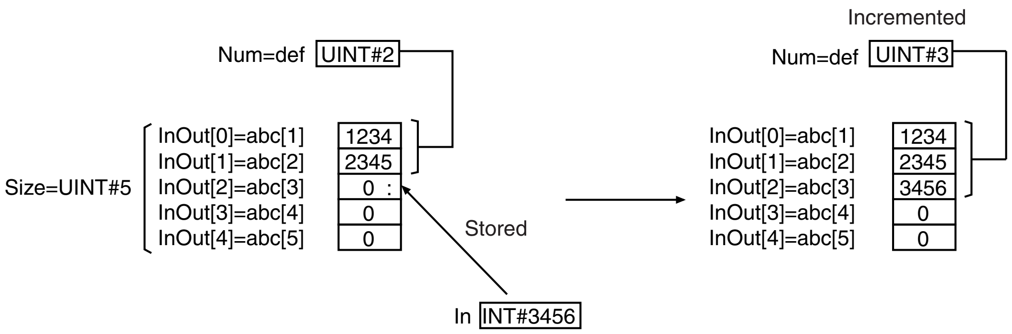
The following example is for when *Size* is UINT#5 and *Num* is UINT#2.

LD



ST

```
StackPush(INT#3456, abc[1], UINT#5, def);
```



Additional Information

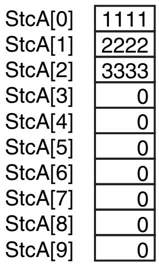
Use the StackFIFO or StackLIFO instruction (page 2-507) to remove the bottom or top value that was stored in the stack.

Precautions for Correct Use

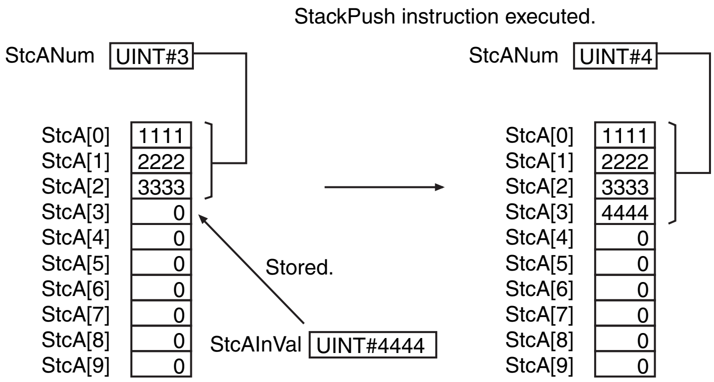
- Use the same data type for *In* and the elements of *InOut[]*. If they are different, a building error will occur.
- When an element in the array is passed to *InOut[]*, all elements below the passed element are processed.
- The value of *InOut[]* or *Num* does not change if the value of *Size* is 0.
- Always use a variable for the input parameter to pass to *In*. A building error will occur if a constant is passed.
- If *In* is an enumeration, you cannot directly pass an enumerator to it. A building error will occur if an enumerator is passed to it directly.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *InOut[]* will not change.
  - The value of *Size* is not 0 and *Num* is greater than or equal to *Size*.
  - The value of *Size* exceeds the array area of *InOut[]*.
  - *In* and *InOut[]* are STRING data and the number of bytes in *In* exceeds the size of *InOut[]*.

Sample Programming

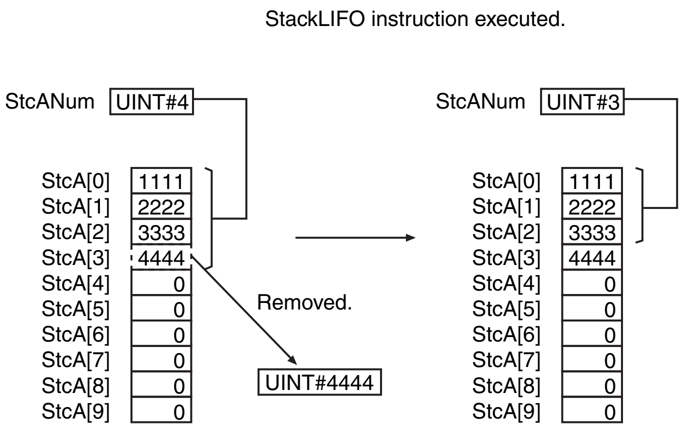
The array variable *StcA[0..9]* is used as a stack. As preparations, three values (UINT#1111, UINT#2222, and UINT#3333) are stored in the stack.



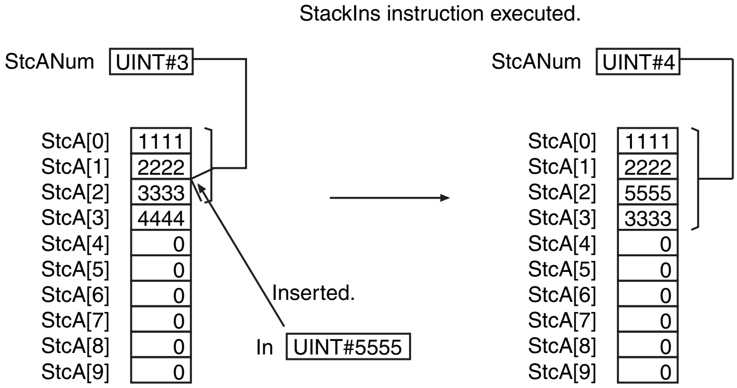
The StackPush instruction is used to store a new value (UINT#4444) at the top of the stack *StcA[3]*. That means there will be four values in the stack.



Then, the StackLIFO instruction is used to remove one value at the top of the stack *StcA[3]*. That means there will be three values in the stack.

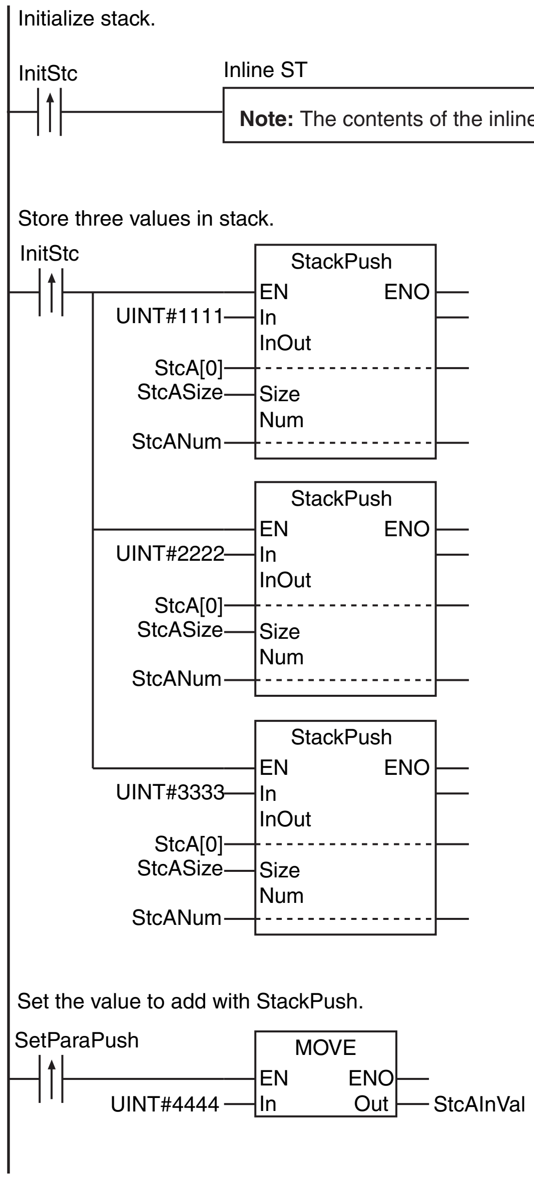


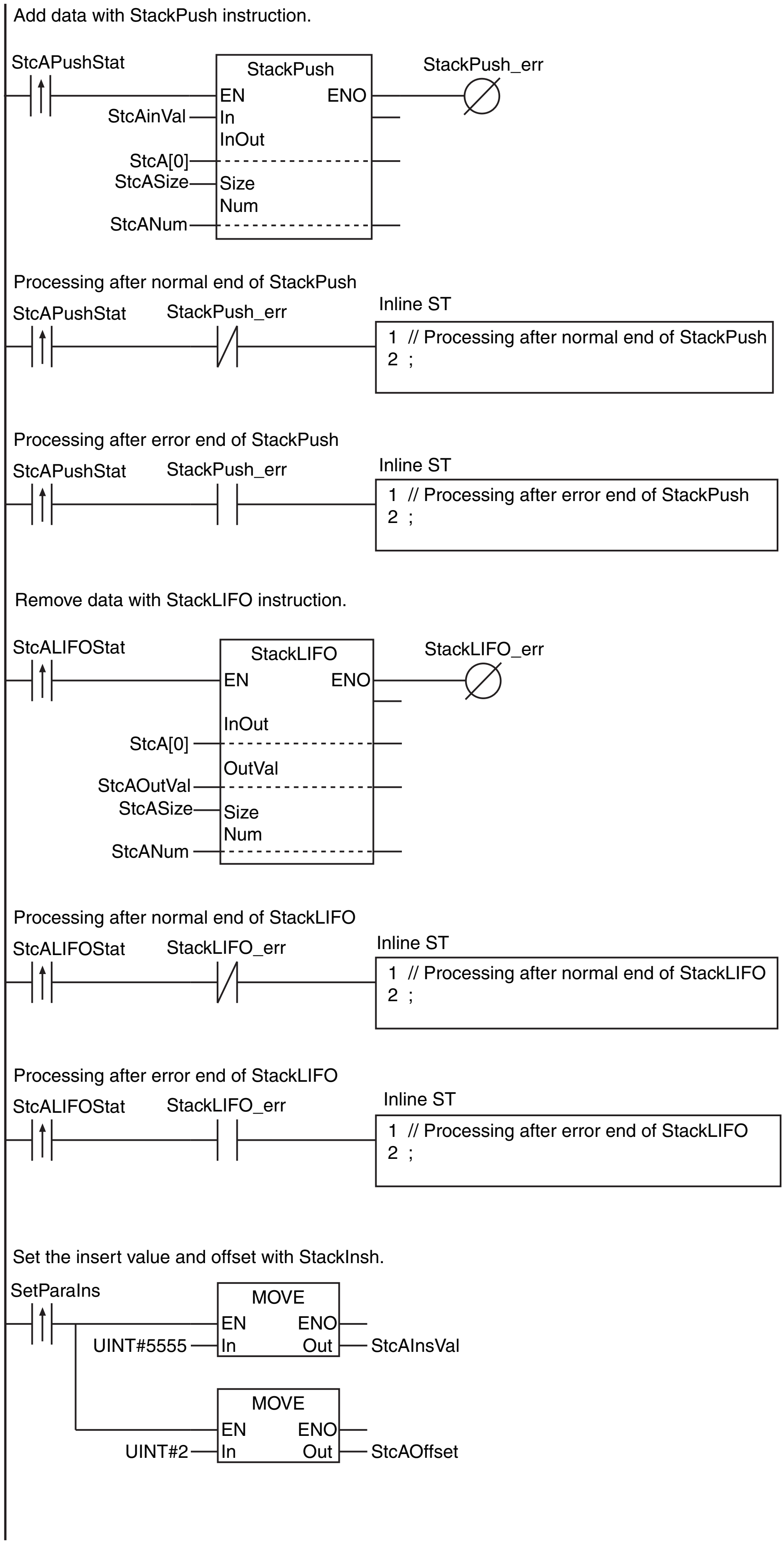
Finally, the StackIns instruction is used to insert a value (UINT#5555) between *StcA[1]* and *StcA[2]*. That means there will be four values in the stack.

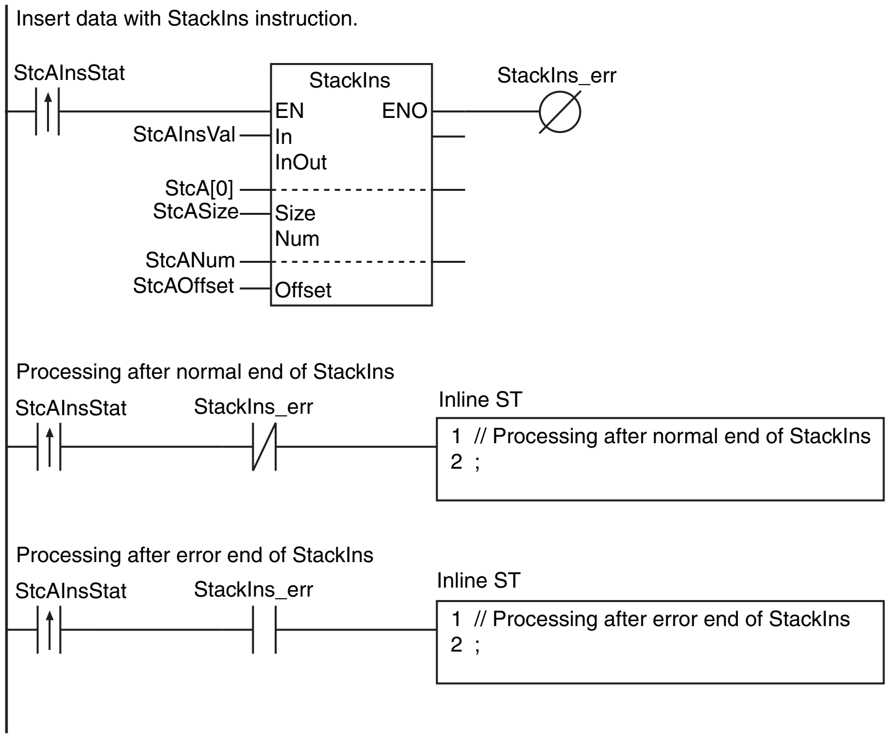


LD

Variable	Data type	Initial value	Comment
InitStc	BOOL	FALSE	Stack initialization condition
StcANum	UINT	0	Number of stored elements
StcA	ARRAY[0..9] OF UINT	[10(0)]	Stack array
StcASize	UINT	0	Number of stack elements
SetParaPush	BOOL	FALSE	Execution condition to set *StcAInVal*.
StcAInVal	UINT	0	Value added by StackPush
StcAPushStat	BOOL	FALSE	StackPush execution condition
StackPush_err	BOOL	FALSE	StackPush error flag
StcALIFOStat	BOOL	FALSE	StackLIFO execution condition
StcAOutVal	UINT	0	Value removed by StackLIFO
StackLIFO_err	BOOL	FALSE	StackLIFO error flag
SetParaIns	BOOL	FALSE	Execution condition to set *StcAInsVal* and *StcAOffset*
StcAInsVal	UINT	0	Value inserted by StackIns
StcAOffset	UINT	0	Offset for StackIns
StcAInsStat	BOOL	FALSE	StackIns execution condition
StackIns_err	BOOL	FALSE	StackIns error flag


Initialize stack.
InitStc
Inline ST
Note: The contents of the inline ST are given below.
Store three values in stack.
InitStc
StackPush
EN
ENO
UINT#1111
In
InOut
StcA[0]
StcASize
Size
Num
StcANum
StackPush
EN
ENO
UINT#2222
In
InOut
StcA[0]
StcASize
Size
Num
StcANum
StackPush
EN
ENO
UINT#3333
In
InOut
StcA[0]
StcASize
Size
Num
StcANum
Set the value to add with StackPush.
SetParaPush
MOVE
EN
ENO
UINT#4444
In
Out
StcAInVal


Add data with StackPush instruction.
StcAPushStat
StackPush
EN
ENO
StackPush_err
StcAinVal
In
InOut
StcA[0]
StcASize
Size
Num
StcANum
Processing after normal end of StackPush
StcAPushStat
StackPush_err
Inline ST
1 // Processing after normal end of StackPush
2 ;
Processing after error end of StackPush
StcAPushStat
StackPush_err
Inline ST
1 // Processing after error end of StackPush
2 ;
Remove data with StackLIFO instruction.
StcALIFOStat
StackLIFO
EN
ENO
StackLIFO_err
InOut
StcA[0]
OutVal
StcAOutVal
StcASize
Size
Num
StcANum
Processing after normal end of StackLIFO
StcALIFOStat
StackLIFO_err
Inline ST
1 // Processing after normal end of StackLIFO
2 ;
Processing after error end of StackLIFO
StcALIFOStat
StackLIFO_err
Inline ST
1 // Processing after error end of StackLIFO
2 ;
Set the insert value and offset with StackInsh.
SetParaIns
MOVE
EN
ENO
UINT#5555
In
Out
StcAInsVal
MOVE
EN
ENO
UINT#2
In
Out
StcAOffset



Contents of Inline ST

```
StcANUM:=0;
Clear(StcA);
StcASize:=SizeOfAry(StcA);
```

ST

Variable	Data type	Initial value	Comment
InitStc	BOOL	FALSE	Stack initialization condition
preInitStc	BOOL	FALSE	Value of *InitStc* from previous task period
StcANum	UINT	0	Number of stored elements
StcA	ARRAY[0..9] OF UINT	[10(0)]	Stack array
StcASize	UINT	0	Number of stack elements
StcAPushStat	BOOL	FALSE	StackPush execution condition
preStcAPushStat	BOOL	FALSE	Value of *StcAPushStat* from previous task period
StcAInVal	UINT	0	Value added by StackPush
StcAPush_OK	BOOL	FALSE	StackPush normal end flag
StcAPushNormalEnd	BOOL	FALSE	Processing after normal end of StackPush
StcAPushErrorEnd	BOOL	FALSE	Processing after error end of StackPush
StcALIFOStat	BOOL	FALSE	StackLIFO execution condition
preStcALIFOStat	BOOL	FALSE	Value of *StcALIFOStat* from previous task period
StcAOutVal	UINT	0	Value removed by StackLIFO
StcALIFO_OK	BOOL	FALSE	StackLIFO normal end flag
StcALIFONormalEnd	BOOL	FALSE	Processing after normal end of StackLIFO
StcALIFOErrorEnd	BOOL	FALSE	Processing after error end of StackLIFO
StcAInsStat	BOOL	FALSE	StackIns execution condition
preStcAInsStat	BOOL	FALSE	Value of *StcAInsStat* from previous task period
StcAInsVal	UINT	0	Value inserted by StackIns
StcAOffset	UINT	0	Offset for StackIns
StcAIns_OK	BOOL	FALSE	StackIns normal end flag
StcAInsNormalEnd	BOOL	FALSE	Processing after normal end of StackIns
StcAInsErrorEnd	BOOL	FALSE	Processing after error end of StackIns

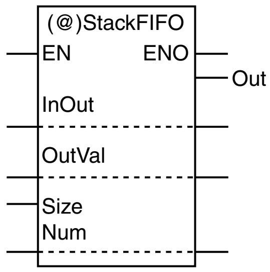
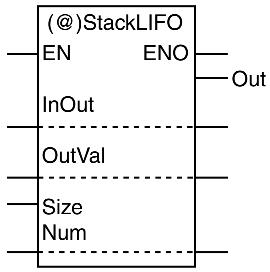
```
// Initialize stack.
IF ( (InitStc=TRUE) AND (preInitStc=FALSE) ) THEN
     StcANum:=0;
     Clear(StcA);
     StcASize:=SizeOfAry(StcA);
END_IF;

// Store three values in stack.
IF ( (InitStc=TRUE) AND (preInitStc=FALSE) ) THEN
     StackPush(In:=UINT#1111, InOut:=StcA[0], Size:=StcASize, Num:=StcANum);
     StackPush(In:=UINT#2222, InOut:=StcA[0], Size:=StcASize, Num:=StcANum);
     StackPush(In:=UINT#3333, InOut:=StcA[0], Size:=StcASize, Num:=StcANum);
END_IF;

preInitStc:=InitStc;

// Add data with StackPush instruction.
IF ( (StcAPushStat=TRUE) AND (preStcAPushStat=FALSE) ) THEN
     StcAInVal:=UINT#4444;
     StackPush(
          In    :=StcAInVal,     // Value to add
          InOut :=StcA[0],       // First element in stack array
          Size  :=StcASize,      // Number of stack elements
          Num   :=StcANum,       // Number of stored elements
          ENO    =>StcAPush_OK); // Normal end flag
```

```
    IF (StcAPush_OK=TRUE) THEN
        StcAPushNormalEnd:=TRUE;  // Processing after normal end
    ELSE
        StcAPushErrorEnd :=TRUE;  // Processing after error end
    END_IF;
END_IF;
preStcAPushStat:=StcAPushStat;

// Remove data with StackLIFO instruction.
IF ( (StcALIFOStat=TRUE) AND (preStcALIFOStat=FALSE) ) THEN
    StackLIFO(
        InOut  :=StcA[0],        // First element in stack array
        OutVal :=StcAOutVal,     // Value removed from stack
        Size   :=StcASize,       // Number of stack elements
        Num    :=StcANum,        // Number of stored elements
        ENO     =>StcALIFO_OK);  // Normal end flag
    IF (StcALIFO_OK=TRUE) THEN
        StcALIFONormalEnd:=TRUE; // Processing after normal end
    ELSE
        StcALIFOErrorEnd:=TRUE;  // Processing after error end
    END_IF;
END_IF;
preStcALIFOStat:=StcALIFOStat;

// Insert data with StackIns instruction.
IF ( (StcAInsStat=TRUE) AND (preStcAInsStat=FALSE) ) THEN
    StcAInsVal:=UINT#5555;
    StcAOffset:=UINT#2;
    StackIns(
        In    :=StcAInsVal,    // Value to insert into stack
        InOut :=StcA[0],       // First element in stack array
        Size  :=StcASize,      // Number of stack elements
        Num   :=StcANum,       // Number of stored elements
        Offset:=StcAOffset,    // Offset at which to insert value
        ENO    =>StcAIns_OK);  // Normal end flag
    IF (StcAIns_OK=TRUE) THEN
        StcAInsNormalEnd:=TRUE;// Processing after normal end
    ELSE
        StcAInsErrorEnd:=TRUE; // Processing after error end
    END_IF;
END_IF;
preStcAInsStat:=StcAInsStat;
```

StackFIFO and StackLIFO

StackFIFO: Removes the bottom value from a stack.

StackLIFO: Removes the top value from a stack.

Instruction	Name	FB/FUN	Graphic expression	ST expression
StackFIFO	First In First Out	FUN		StackFIFO(InOut, OutVal, Size, Num);
StackLIFO	Last In First Out	FUN		StackLIFO(InOut, OutVal, Size, Num);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Size	Number of stack elements	Input	Number of stack array elements	Depends on data type.	---	1
InOut[] (array)	Stack array		Array that functions as stack			
OutVal	Output value	In-out	Value or structure output from stack	Depends on data type.	---	---
Num	Number of stored elements		Number of elements stored in stack			
Out	Return value	Output	Always TRUE	TRUE only	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Size							OK													
InOut[] (array)	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	Arrays of enumerations or structures can also be specified.																			
OutVal	Must be the same data type as the elements of *InOut[]*.																			
Num							OK													
Out	OK																			

Function

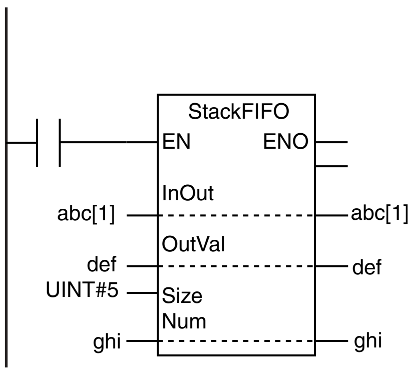
The instruction assumes that there are number of stored elements *Num* elements stored in stack array *InOut[]*. The instruction removes a value from the stack and assigns it to output value *OutVal*. For number of stack elements *Size*, specify the number of elements in *InOut[]* to use as a stack.

● StackFIFO

The StackFIFO removes the bottom value from a stack. Value of *InOut[0]* is assigned to *OutVal*. Then, all *Num* – 1 elements from *InOut[1]* are shifted to the next lower element in the stack array. Then 0 is stored in *InOut[Num–1]*. Finally, *Num* is decremented.

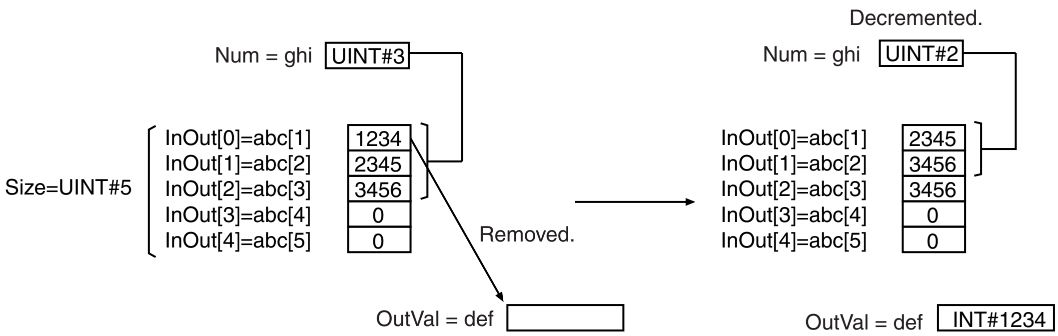
The following example is for when *Size* is UINT#5 and *Num* is UINT#3.

LD



ST

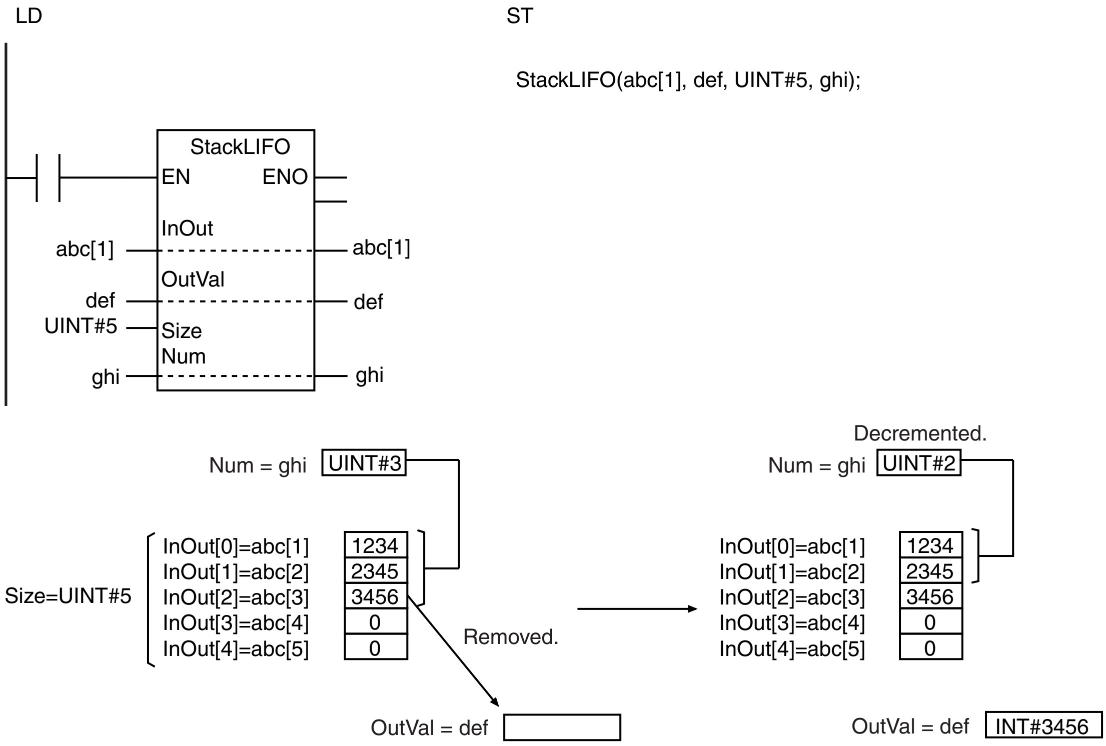
```
StackFIFO(abc[1], def, UINT#5, ghi);
```



● StackLIFO

The StackLIFO instruction removes the top value from a stack. Value of *InOut[Num–1]* is assigned to *OutVal.* Then, *Num* is decremented.

The following example is for when *Size* is UINT#5 and *Num* is UINT#2.



Precautions for Correct Use

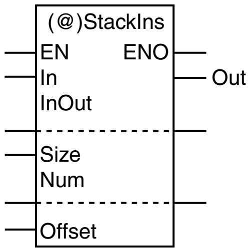
- Use the same data type for *InOut[]* and *OutVal. If they are different, a building error will occur.*
- When an element in the array is passed to *InOut[]*, all elements below the passed element are processed.
- The values in *InOut[]*, *Num*, and *OutVal* do not change if the value of *Size* or *Num* is 0.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *OutVal* will not change.
  - The values of *Num* and *Size* are not 0 and *Num* is greater than *Size*.
  - The value of *Size* exceeds the array area of *InOut[]*.
  - *InOut[]* is a STRING array and any of the elements does not end in a NULL character.
  - *InOut[]* is a STRING array and the number of bytes in the elements exceeds the size of *OutVal*.

Sample Programming

Refer to the sample programming that is provided for the StackPush instruction (page 2-498).

StackIns

The StackIns instruction inserts a value at a specified position in a stack.

Instruction	Name	FB/FUN	Graphic expression	ST expression
StackIns	Insert into Stack	FUN		StackIns(In, InOut, Size, Num, Offset);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Insert value	Input	Value, structure, or structure member to insert into the stack	Depends on data type.	---	*
Size	Number of stack elements		Number of stack array elements			1
Offset	Offset		Position in stack at which to insert *In*			0
InOut[] (array)	Stack array	In-out	Array that functions as stack	Depends on data type.	---	---
Num	Number of stored elements		Number of elements stored in stack			
Out	Return value	Output	Always TRUE	TRUE only	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

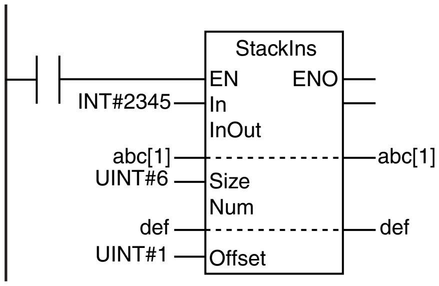
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	An enumeration, structure, or structure member can also be specified.																			
Size							OK													
Offset							OK													
InOut[] (array)	Must be an array with elements that have the same data type as *In*.																			
Num							OK													
Out	OK																			

Function

The instruction assumes that there are number of stored elements *Num* elements stored in stack array *InOut[]*. Insert value *In* is inserted at the position specified by the offset *Offset* (*InOut[Offset]*). All higher elements, i.e., *InOut[Offset]* to *InOut[Num*–1*]*, are moved to the next higher element in the stack array. Then, *Num* is incremented. For number of stack elements *Size*, specify the number of elements in *InOut[]* to use as a stack.

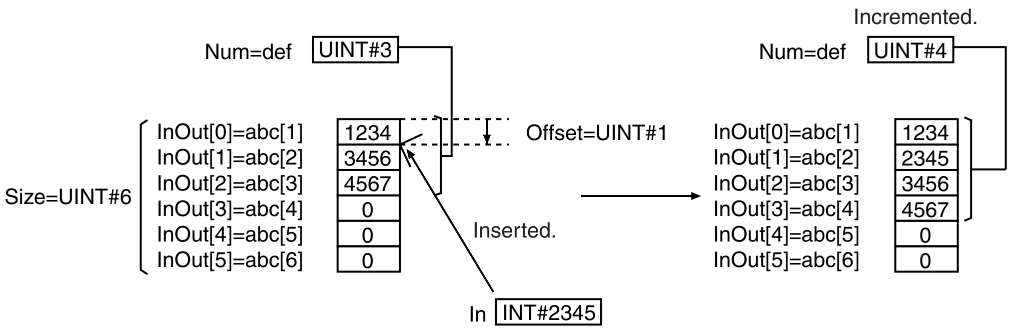
The following example is for when *Size* is UINT#6, *Num* is UINT#3 and *Offset* is UINT#1.

LD



ST

StackIns(INT#2345, abc[1], UINT#6, def, UINT#1);



Precautions for Correct Use

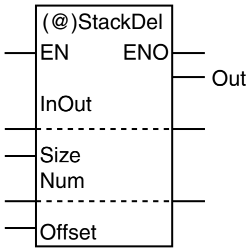
- Use the same data type for *In* and *InOut[]*. If they are different, a building error will occur.
- When an element in the array is passed to *InOut[]*, all elements below the passed element are processed.
- The values in *InOut[]* and *Num* do not change if the value of *Size* is 0.
- Always use a variable for the input parameter to pass to *In*. A building error will occur if a constant is passed.
- If *In* is an enumeration, you cannot directly pass an enumerator to it. A building error will occur if an enumerator is passed to it directly.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *InOut[]* will not change.
  - The value of *Size* is not 0 and *Size* is not greater than *Num* and *Num* is not greater than or equal to *Offset*.
  - The value of *Size* exceeds the array area of *InOut[]*.
  - *In* and *InOut[]* are STRING data and the number of bytes in *In* exceeds the size of *InOut[]*.

Sample Programming

Refer to the sample programming that is provided for the StackPush instruction (page 2-498).

StackDel

The StackDel instruction deletes a value from a specified position in a stack.

Instruction	Name	FB/FUN	Graphic expression	ST expression
StackDel	Delete from Stack	FUN		StackDel(InOut, Size, Num, Offset);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Size	Number of stack elements	Input	Number of stack array elements	Depends on data type.	---	1
Offset	Offset		Offset of value to delete from stack			0
InOut[] (array)	Stack array	In-out	Array that functions as stack	Depends on data type.	---	---
Num	Number of stored elements		Number of elements stored in stack			
Out	Return value	Output	Always TRUE	TRUE only	---	---

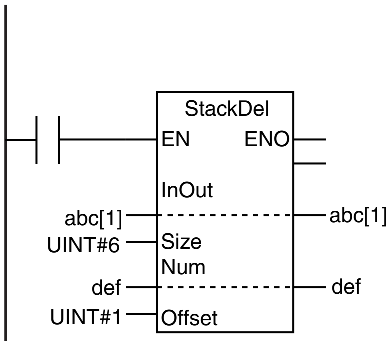
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Size							OK													
Offset							OK													
InOut[] (array)	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	Arrays of enumerations or structures can also be specified.																			
Num							OK													
Out	OK																			

Function

The instruction assumes that there are number of stored elements *Num* elements stored in stack array *InOut[]*. The value is deleted from the position specified by the offset *Offset* (*InOut[Offset]*). All higher elements, i.e., *InOut[Offset*+1*]* to *InOut[Num*–1*]*, are moved to the next lower element in the stack array. Then, *Num* is decremented. For number of stack elements *Size*, specify the number of elements in *InOut[]* to use as a stack.

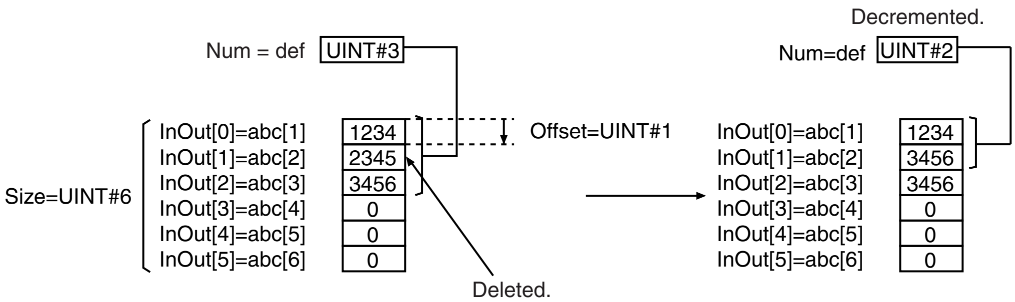
The following example is for when *Size* is UINT#6, *Num* is UINT#3 and *Offset* is UINT#1.

LD



ST

```
StackDel(abc[1], UINT#6, def, UINT#1);
```

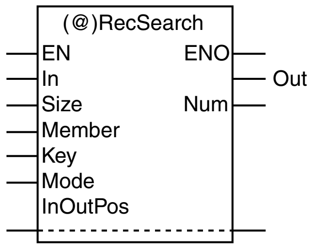


Precautions for Correct Use

- When an element in the array is passed to *InOut[]*, all elements below the passed element are processed.
- The values in *InOut[]* and *Num* do not change if the value of *Size* or *Num* is 0.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *InOut[]* will not change.
  - The values of *Num* and *Size* are not 0 and *Size* is not greater than or equal to *Num* and *Num* is not greater than *Offset*.
  - The value of *Size* exceeds the array area of *InOut[]*.

RecSearch

The RecSearch instruction searches an array of structures for elements that match the search key with the specified method.

Instruction	Name	FB/FUN	Graphic expression	ST expression
RecSearch	Record Search	FUN		Out:=RecSearch(In, Size, Member, Key, Mode, InOutPos, Num);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array to search	Input	Array of structures to search	---	---	*
Size	Number of elements to search		Number of array elements to search	Depends on data type.		1
Member	Member to search		Member of In[] structure to search			*
Key	Search key		Search value			
Mode	Search method		Search method	_LINEAR, _BIN_ASC, _BIN_DESC		_LINEAR
InOutPos[] (array)	Element numbers of matching elements	In-out	Element numbers of matching elements	Depends on data type.	---	---
Out	Search result	Output	TRUE: There are elements that match conditions FALSE: There are no elements that match conditions	Depends on data type.	---	---
Num	Number of matches		Number of matches			

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)	Specify an array of structures.																			
Size							OK													
Member						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
																*				
	Specify the same data type as the search member of *In[]*																			
Key	Must be the same data type as *Member*.																			
Mode	Refer to *Function* for the enumerators of the enumerated type _eSEARCH_MODE.																			
InOutPos[] (array)							OK													
Out	OK																			
Num							OK													

* You can specify TIME, DATE, TOD, DT, and STRING data with CPU Units with unit version 1.01 or later and Sysmac Studio version 1.02 or higher.

Function

The RecSearch instruction searches *Size* elements in the array of structures *In[]*. The search range is therefore from *In[0]* to *In[Size−1]*. The instruction searches member to search Member in the structures for members that match the search key *Key*.
One of the members to search in the elements of *In[]* is passed as an argument to *Member*.
If any matching elements are found, the value of search result *Out* changes to TRUE. The element number of the matching element is assigned to *InOutPos[0]* and the number of matching elements is assigned to *Num*. If there is more than one matching element, the element number of the lowest matching element in *In[]* is assigned to *InOutPos[0]*. If there are no matching elements, the value of *Out* will be FALSE and *InOutPos[0]* and *Num* will be 0.

Always attach the element number to input parameter that is passed to *In[]*, e.g., *array[3]*.

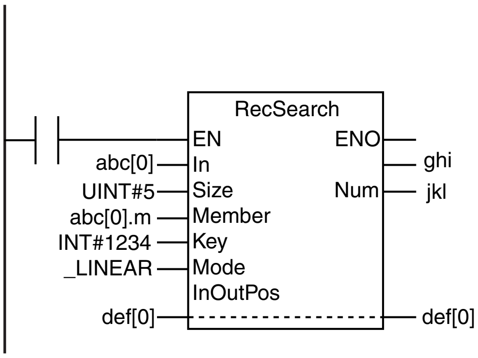
The data type of search method *Mode* is enumerated type _eSEARCH_MODE. The meanings of the enumerators are as follows:

Enumerator	Meaning
_LINEAR	Linear search
_BIN_ASC	Ascending binary search
_BIN_DESC	Descending binary search

For a linear search, the search is performed in order from the first element of *In[]*.

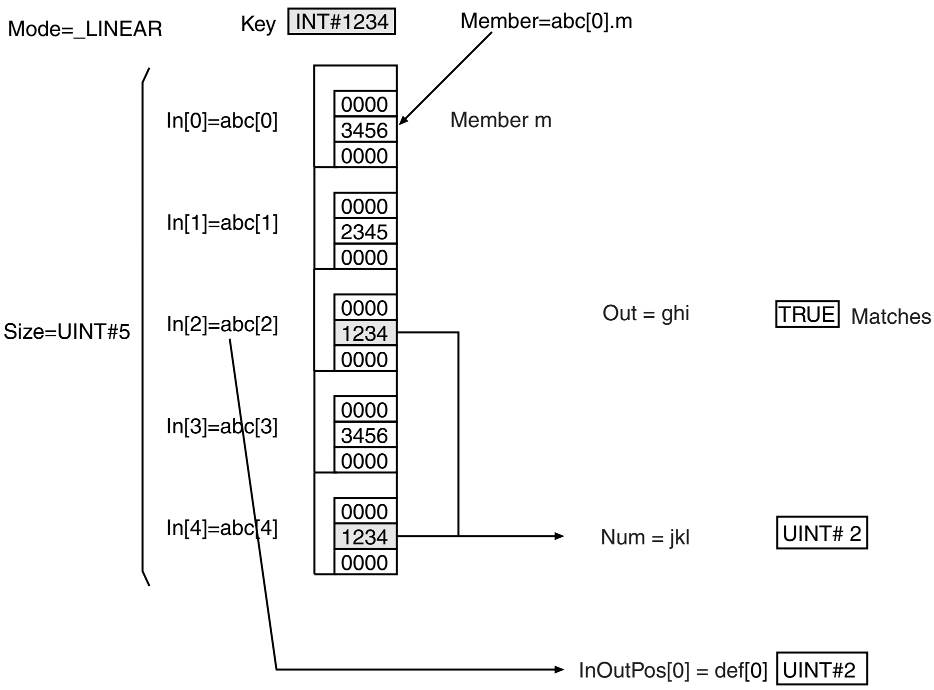
The following example is for when *Size* is UINT#5, *Key* is INT#1234 and *Mode* is _LINEAR.

LD



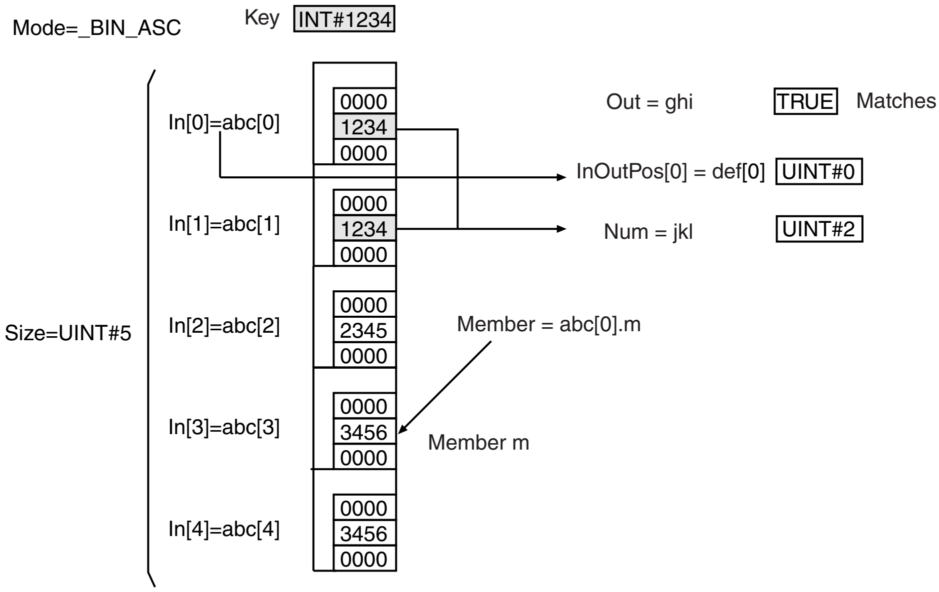
ST

```
ghi:=RecSearch(abc[0], UINT#5, abc[0].m,
        INT#1234, _LINEAR, def[0], jkl);
```



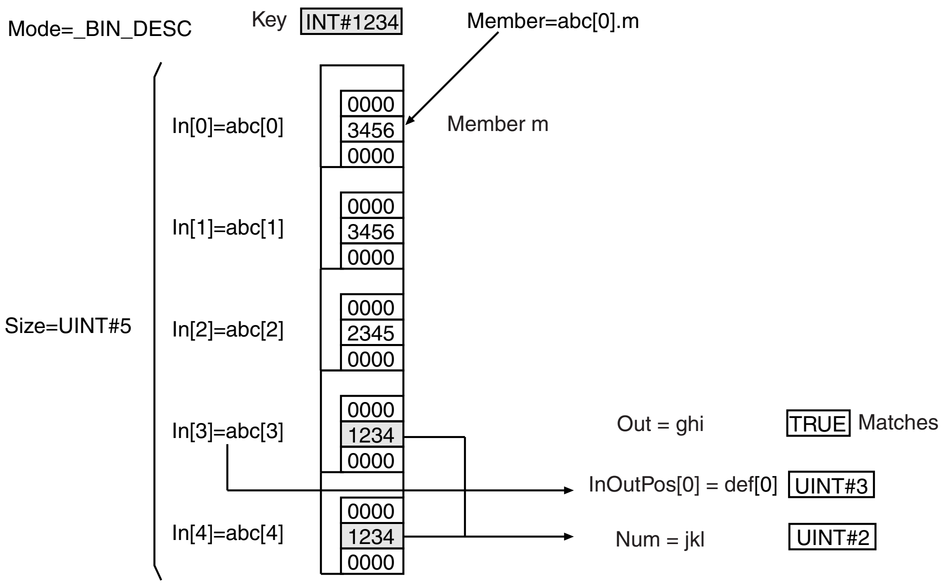
For an ascending binary search, the array elements in the input parameter that is passed to *In[]* must be in ascending order before this instruction is executed. Then a binary search is performed by executing this instruction.

Using the same example as before, the order of the array elements and the processing results will be as shown below for an ascending binary search.



For a descending binary search, the array elements in the input parameter that is passed to *In[]* must be in descending order before this instruction is executed. Then a binary search is performed by executing this instruction.

Using the same example as before, the order of the array elements and the processing results will be as shown below for a descending binary search.



Additional Information

- *In[]* can be a member of a higher-level structure.

  Example: In[0]=str0.str1[0]
- *In[]* can be an array with two or more dimensions. If *In[]* is a two-dimensional array, the element number in the first dimension of the element that matches the search conditions is assigned to *InOutPos[0]* and the element number in the second dimension is assigned to *InOutPos[1]*.

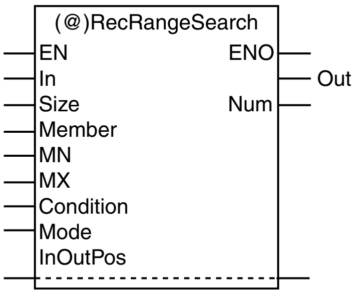
- If *In[]* is a three-dimensional array, the element number in the first dimension of the element that matches the search conditions is assigned to *InOutPos[0]*, the element number in the second dimension is assigned to *InOutPos[1]*, and the element number in the third dimension is assigned to *InOutPos[2]*.
- When you search TIME, DT, or TOD data, adjust the data so that the precision of *Member* and *Key* is the same. Use the following instructions to adjust the precision of the values: TruncTime (page 2-657), TruncDt (page 2-661), and TruncTod (page 2-665).

Precautions for Correct Use

- Use an array that is the element of a structure for *In[]*. Otherwise, a building error will occur.
- The data types of *Key* and *Member* must be the same. If they are different, a building error will occur.
- When an element in the array is passed to *In[]*, all elements below the passed element are processed.
- If *Member* is a real number, depending on the value of *Member*, the desired results may not be achieved due to error.
- If *Key* is a real number, do not specify nonnumeric data for *Key*.
- If the value of *Size* is 0, the value of *Out* is FALSE and the value of *Num* is 0. *InOutPos[]* does not change.
- The correct result is not obtained if the value of *Mode* is _BIN_ASC or _BIN_DESC and the elements of *In[]* are not in ascending or descending order. Place the elements in ascending or descending order before executing this instruction.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out*, *InOutPos[]*, and *Num* will not change.
  - The value of *Mode* is outside of the valid range.
  - The value of *Size* exceeds the array area of *In[]*.
  - *Member* is not a member of *In[]*.
  - The array size of *InOutPos[]* is smaller than the number of dimensions of *In[]*.
  - *Member* is STRING data and it does not end in a NULL character.

RecRangeSearch

The RecRangeSearch instruction searches an array of structures for elements that match the search condition range with the specified method.

Instruction	Name	FB/FUN	Graphic expression	ST expression
RecRangeSearch	Range Record Search	FUN		Out:=RecRangeSearch(In, Size, Member, MN, MX, Condition, Mode, InOutPos, Num);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array to search	Input	Array of structures to search	---	---	*
Size	Number of elements to search		Number of array elements to search	Depends on data type.		1
Member	Member to search		Member of *In[]* structure to search			*
MN	Search condition lower limit		Search condition lower limit			
MX	Search condition upper limit		Search condition upper limit			
Condition	Search condition		Search condition	_EQ_BOTH, _EQ_MIN, _EQ_MAX, _NE_BOTH		_EQ_ BOTH
Mode	Search method		Search method	_LINEAR, _BIN_ASC, _BIN_DESC		_LIN- EAR
InOutPos[] (array)	Element numbers of matching elements	In-out	Element numbers of matching elements	Depends on data type.	---	---
Out	Search result	Output	TRUE: There are elements that match conditions FALSE: There are no elements that match conditions	Depends on data type.	---	---
Num	Number of matches		Number of matches			

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)	Specify an array of structures.																			
Size							OK													
Member						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
																*				
	Specify the same data type as the search member of *In[].*																			
MN	Must be the same data type as *Member*.																			
MX	Must be the same data type as *Member*.																			
Condition	Refer to *Function* for the enumerators for the enumerated type _eSEARCH_CONDITION.																			
Mode	Refer to *Function* for the enumerators for the enumerated type _eSEARCH_MODE.																			
InOutPos[] (array)							OK													
Out	OK																			
Num							OK													

* You can specify TIME, DATE, TOD, DT, and STRING data with CPU Units with unit version 1.01 or later and Sysmac Studio version 1.02 or higher.

Function

The RecRangeSearch instruction searches Size elements in the array of structures *In[]*. The search range is therefore from *In[0]* to *In[Size−1]*. The instruction searches member to search *Member* in the structures for members that match the search condition.

Condition specifies the search condition. *Mode* specifies the search method. Details are provided below. One of the members to search in the elements of *In[]* is passed as an argument to *Member*.

If any elements that match the search condition are found, the value of search result *Out* changes to TRUE. The element number of the matching element is assigned to *InOutPos[0]* and the number of matching elements is assigned to *Num*. If there is more than one matching element, the element number of the lowest matching element in *In[]* is assigned to *InOutPos[0]*. If there are no matching elements, the value of *Out* will be FALSE and *InOutPos[0]* and *Num* will be 0.

Always attach the element number to input parameter that is passed to *In[]*, e.g., *array[3]*.

The data type of search condition *Condition* is enumerated type _eSEARCH_CONDITION. The meanings of the enumerators are as follows:

Enumerator	Meaning
_EQ_BOTH	MN ≤ Member ≤ MX
_EQ_MIN	MN ≤ Member < MX
_EQ_MAX	MN < Member ≤ MX
_NE_BOTH	MN < Member < MX

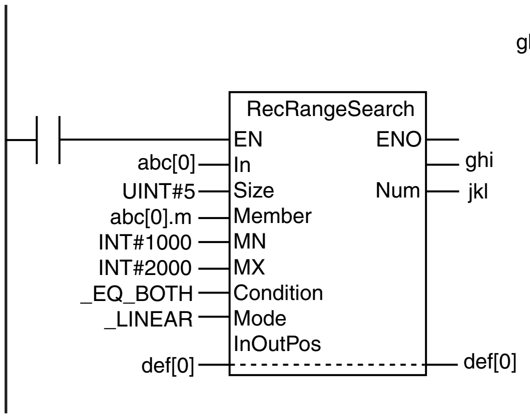
The data type of search method *Mode* is enumerated type _eSEARCH_MODE. The meaning of the enumerators are as follows:

Enumerator	Meaning
_LINEAR	Linear search
_BIN_ASC	Ascending binary search
_BIN_DESC	Descending binary search

For a linear search, the search is performed in order from the first element of *In[].*

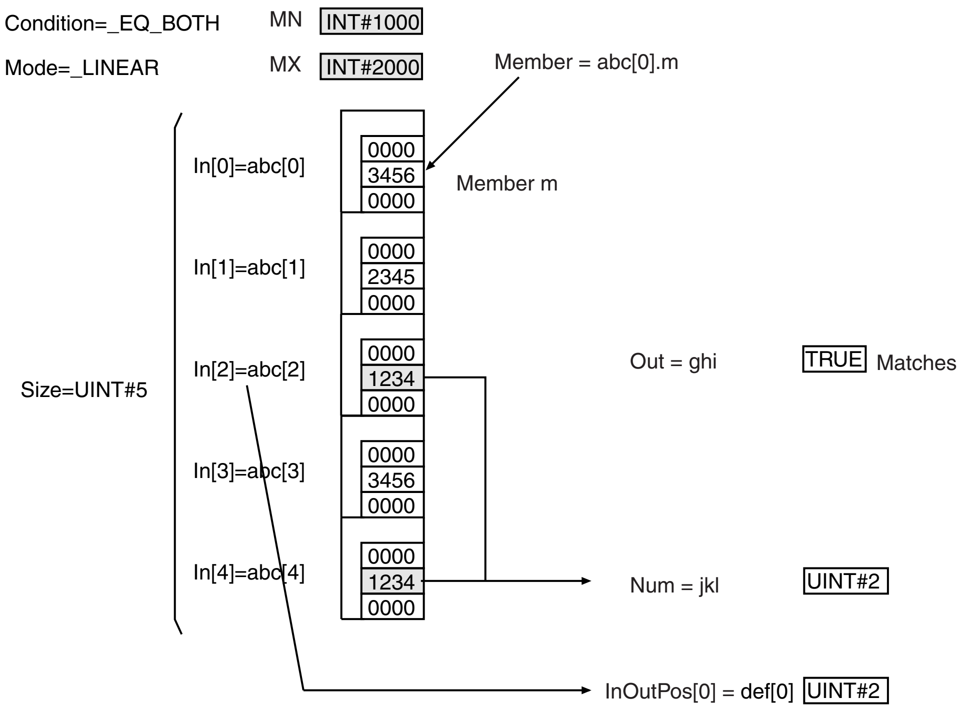
The following example is for when *Size* is UINT#5, *MN* is INT#1000, *MX* is INT#2000, *Condition* is _EQ_BOTH and *Mode* is _LINEAR.

LD



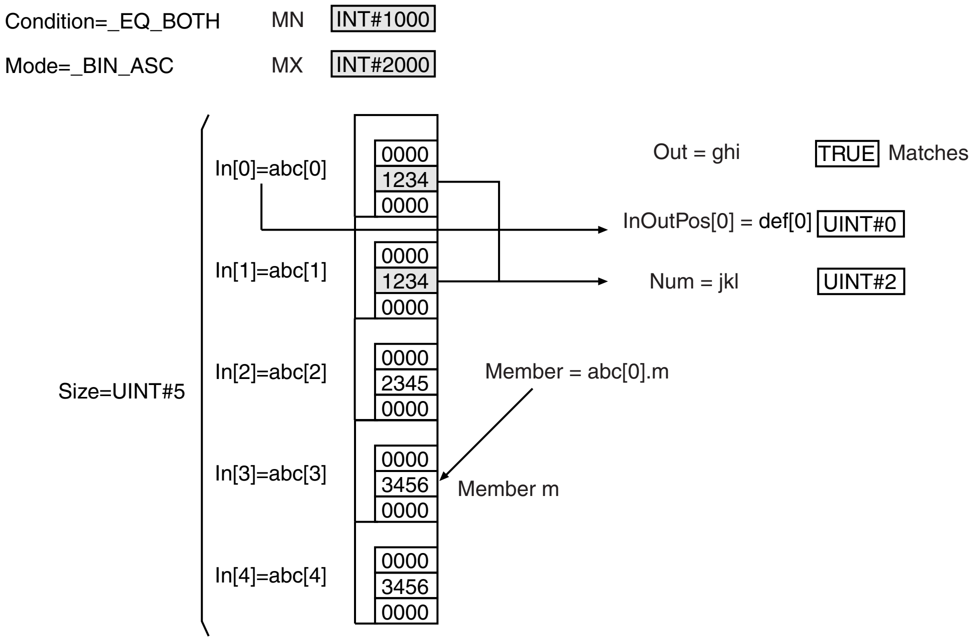
ST

```
ghi:=RecRangeSearch(abc[0], UINT#5, abc[0].m, INT#1000,
                    INT#2000, _EQ_BOTH, _LINEAR, def[0], jkl);
```



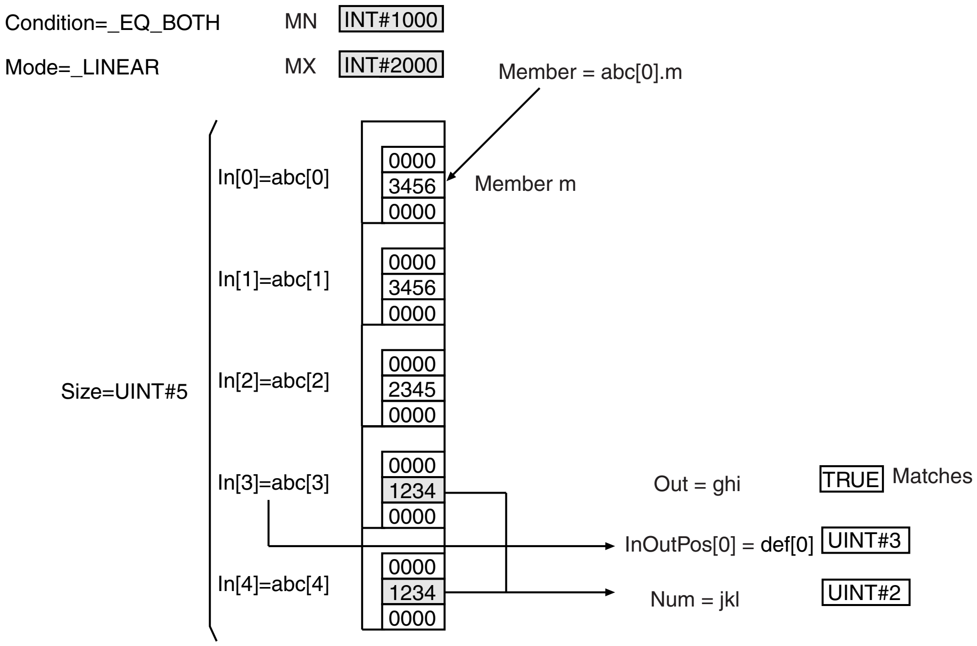
For an ascending binary search, the array elements in the input parameter that is passed to *In[]* must be in ascending order before this instruction is executed. Then a binary search is performed by executing this instruction.

Using the same example as before, the order of the array elements and the processing results will be as shown below for an ascending binary search.



For a descending binary search, the array elements in the input parameter that is passed to *In[]* must be in descending order before this instruction is executed. Then a binary search is performed by executing this instruction.

Using the same example as before, the order of the array elements and the processing results will be as shown below for a descending binary search.



Additional Information

- *In[]* can be a member of a higher-level structure.

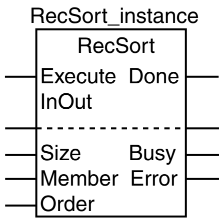
  Example: In[0]=str0.str1[0]
- *In[]* can be an array with two or more dimensions. If *In[]* is a two-dimensional array, the element number in the first dimension of the element that matches the search conditions is assigned to *InOutPos[0]* and the element number in the second dimension is assigned to *InOutPos[1]*.
- If *In[]* is a three-dimensional array, the element number in the first dimension of the element that matches the search conditions is assigned to *InOutPos[0]*, the element number in the second dimension is assigned to *InOutPos[1]*, and the element number in the third dimension is assigned to *InOutPos[2]*.
- When you search TIME, DT, or TOD data, adjust the data so that the precision of *Member*, *MN*, and *MX* is the same. Use the following instructions to adjust the precision of the values: TruncTime (page 2-657), TruncDt (page 2-661), and TruncTod (page 2-665).

Precautions for Correct Use

- Make the data types of *Member*, *MN*, and *MX* the same as the data type of the members that are searched in *In[]*. Otherwise, a building error will occur.
- Use an array that is the element of a structure for *In[].* Otherwise, a building error will occur.
- When an element in the array is passed to *In[]*, all elements below the passed element are processed.
- If *Member* is a real number, depending on the value of *Member*, the desired results may not be achieved due to error.
- If *MN* or *MX* is a real number, do not specify nonnumeric data for *MN* or *MX*.
- If the value of *Size* is 0, the value of *Out* is FALSE and the value of *Num* is 0. *InOutPos[]* does not change.
- The correct result is not obtained if the value of *Mode* is _BIN_ASC or _BIN_DESC and the elements of *In[]* are not in ascending or descending order. Place the elements in ascending or descending order before executing this instruction.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out*, *InOutPos[]*, and *Num* will not change.
  - *MN* is greater than *MX*.
  - The value of *Condition* is outside of the valid range.
  - The value of *Mode* is outside of the valid range.
  - The value of *Size* exceeds the array area of *In[]*.
  - *Member* is not a member of *In[]*.
  - The array size of *InOutPos[]* is smaller than the number of dimensions of *In[]*.

RecSort

The RecSort instruction sorts the elements of an array of structures.

Instruction	Name	FB/FUN	Graphic expression	ST expression
RecSort	Record Sort	FB		RecSort_instance(Execute, InOut, Size, Member, Order, Done, Busy, Error);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Size	Number of elements to sort	Input	Number of array elements to sort	Depends on data type.	---	1
Member	Member to sort		Member of I*n[]* structure to sort			*
Order	Sort order		Sort order	_ASC, _DESC		_ASC
InOut[] (array)	Sort array	In-out	Array of structures to sort	---	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Size							OK													
Member						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
																*				
	Specify the same data type as the sort member of *InOut[]*																			
Order	Refer to *Function* for the enumerators of the enumerated type _eSORT_ORDER.																			
InOut[] (array)	Specify an array of structures.																			

* You can specify TIME, DATE, TOD, DT, and STRING data with CPU Units with unit version 1.01 or later and Sysmac Studio version 1.02 or higher.

Function

When the value of *Execute* is TRUE, *Size* elements of *InOut[]* (a structure array) is sorted. Specifically, the elements from *InOut[0]* to *InOut[Size−1]* are sorted. Specifically, the elements from *InOut[0]* to *InOut[Size−1]* are sorted. *Order* specifies the sort order. Details are provided below. One of the members to sort in the elements of *In[]* is passed as an argument to *Member*.

Always attach the element number to the in-out parameter that is passed to *InOut[]*, e.g., *array[3]*.

The data type of sort order *Order* is enumerated type _eSORT_ORDER. The meaning of the enumerators are as follows:

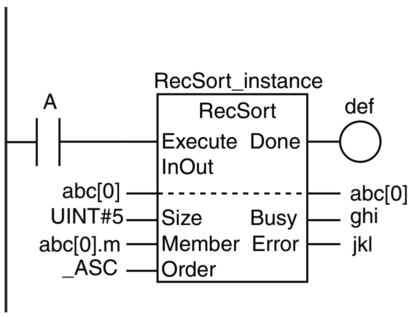
Enumerator	Meaning
_ASC	Ascending
_DESC	Descending

The relationship between values with data types that are not integers or real numbers are determined as given in the following table.

Data type	Relationship
TIME	The numerically larger value is considered to be larger.
DATE, TOD, or DT	Later dates or times of day are considered to be larger.
STRING	The specifications are the same as for the LTascii, LEascii, GTascii, and GEascii instructions (page 2-104). Refer to the specified page for details.

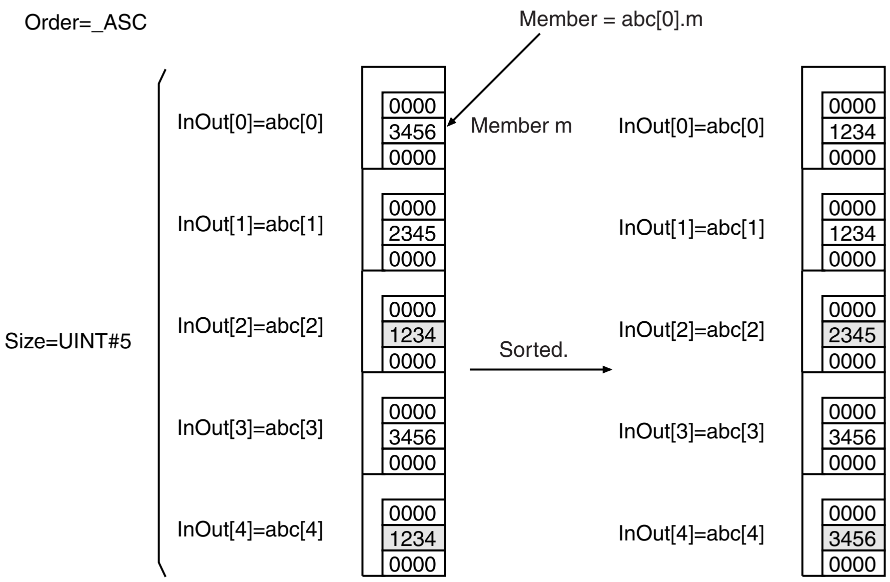
The following example is for when *Size* is UINT#5, *Member* is 3456 and *Order* is *_Asc*.

LD



ST

```
RecSort_instance(A, abc[0], UINT#5, abc[0].m, _ASC, def, ghi, jkl);
```



Additional Information

- If the power supply is interrupted during execution of this instruction, the contents of *InOut* may be corrupted. If you back up the contents of *InOut[]* each time the instruction is completed normally, you can restore the data if it is corrupted. Refer to *Sample Programming*.
- When you sort TIME, DT, or TOD data, adjust the data so that the precision of *Member* is the same. Use the following instructions to adjust the precision of the values: TruncTime (page 2-657), TruncDt (page 2-661), and TruncTod (page 2-665).

Precautions for Correct Use

- Use an array that is the element of a structure for *InOut[]*. Otherwise, a building error will occur.
- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to *FALSE* or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- If *Member* is a real number, depending on the value of *Member*, the desired results may not be achieved due to error.
- When an element in the array is passed to *InOut[]*, all elements below the passed element are processed.
- If the value of *Size* is 0, the value of *Done* will be TRUE and *InOut[]* will not change.
- An error occurs in the following cases. *Done* and *Busy* will be FALSE and *Error* will be TRUE.
  - The value of *Order* is outside of the valid range.
  - The value of *Size* exceeds the array area of *InOut[]*.
  - *Member* is not a member of *InOut[]*.
  - *Member* is STRING data and it does not end in a NULL character.

Sample Programming

In this sample, the RecSort instruction is used to sort an array *Abc[]* of *MyStr* structures in ascending order. The member to sort is *Abc[].m*. To prevent loosing data even if power is interrupted during processing, *Abc[]* is backed up in a variable named *Abc_backup[]* before sorting. If a power interruption occurs, the contents of *Abc_backup[]* is restored to *Abc[]* and the sort operation is redone.

Definitions of Global Variables

Data Types

Variable	Data type	Comment
MyStr	STRUCT	Structure
l	BOOL	Member
m	INT	Member
n	REAL	Member

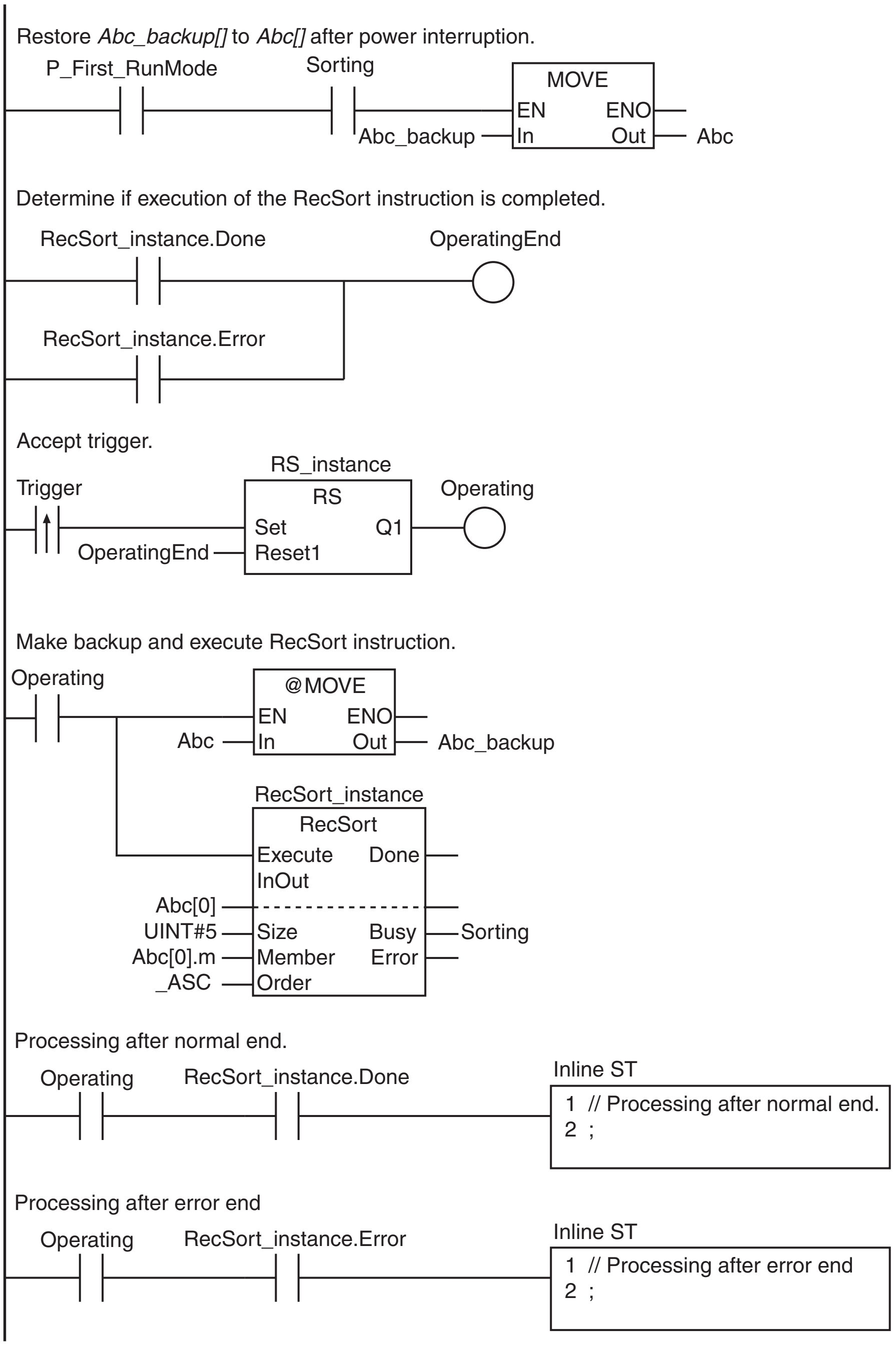
Global Variables

Variable	Data type	Initial value	Retain	Comment
Abc	ARRAY[0..4] OF MyStr	[5((l:=FALSE, m:=0, n:=0.0))]	✔	Sort array
Abc_backup	ARRAY[0..4] OF MyStr	[5((l:=FALSE, m:=0, n:=0.0))]	✔	Backup of *Abc[]*

LD

Internal Variables	Variable	Data type	Initial value	Retain	Comment
	Sorting	BOOL	FALSE	✔	Processing (retained)
	OperatingEnd	BOOL	FALSE		Processing completed
	Trigger	BOOL	FALSE		Execution condition
	Operating	BOOL	FALSE		Processing
	RS_instance	RS			
	RecSort_instance	RecSort			

External Variables	Variable	Data type	Comment
	Abc	ARRAY[0..4] OF MyStr	Sort array
	Abc_backup	ARRAY[0..4] OF MyStr	Backup of *Abc[]*



Restore *Abc_backup[]* to *Abc[]* after power interruption.
P_First_RunMode
Sorting
MOVE
EN
ENO
Abc_backup
In
Out
Abc
Determine if execution of the RecSort instruction is completed.
RecSort_instance.Done
OperatingEnd
RecSort_instance.Error
Accept trigger.
RS_instance
Trigger
RS
Operating
Set
Q1
OperatingEnd
Reset1
Make backup and execute RecSort instruction.
Operating
@MOVE
EN
ENO
Abc
In
Out
Abc_backup
RecSort_instance
RecSort
Execute
Done
InOut
Abc[0]
UINT#5
Size
Busy
Sorting
Abc[0].m
Member
Error
_ASC
Order
Processing after normal end.
Operating
RecSort_instance.Done
Inline ST
1 // Processing after normal end.
2 ;
Processing after error end
Operating
RecSort_instance.Error
Inline ST
1 // Processing after error end
2 ;

ST

Internal Variables	Variable	Data type	Initial value	Retain	Comment
	Sorting	BOOL	FALSE	✔	Processing (retained)
	Trigger	BOOL	FALSE		Execution condition
	LastTrigger	BOOL	FALSE		Value of *Trigger* from previous task period
	OperatingStart	BOOL	FALSE		Processing started
	Operating	BOOL	FALSE		Processing
	RS_instance	RS			
	RecSort_instance	RecSort			

External Variables	Variable	Data type	Comment
	Abc	ARRAY[0..4] OF MyStr	Sort array
	Abc_backup	ARRAY[0..4] OF MyStr	Backup of *Abc[]*

```
// Restore Abc_backup[] to Abc[] after power interruption.
IF ( (P_First_RunMode = TRUE) AND (Sorting = TRUE) ) THEN
    Abc:=Abc_backup;
END_IF;

// Detect when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) ) THEN
    OperatingStart :=TRUE;
    Operating      :=TRUE;
END_IF;
LastTrigger:=Trigger;

// Initialize RecSort instruction.
IF (OperatingStart=TRUE) THEN
    Abc_backup:=Abc;
    RecSort_instance(
        Execute  :=FALSE,     // Start condition
        InOut    :=Abc[0],    // Sort array
        Member   :=Abc[0].m); // Member to sort
    OperatingStart:=FALSE;
END_IF;

// Execute RecSort instruction.
IF (Operating=TRUE) THEN
    RecSort_instance(
        Execute:=TRUE,
        InOut  :=Abc[0],
        Size   :=UINT#5,
        Member :=Abc[0].m,
        Order  :=_ASC,
        Busy    =>Sorting);

    IF (RecSort_instance.Done=TRUE) THEN
        // Processing after normal end.
        Operating:=FALSE;
    END_IF;

    IF (RecSort_instance.Error=TRUE) THEN
        // Processing after error end.
        Operating:=FALSE;
    END_IF;
END_IF;
```

RecNum

The RecNum instruction finds the number of records in an array of structures to the end data.

Instruction	Name	FB/FUN	Graphic expression	ST expression
RecNum	Get Number of Records	FUN	(@)RecNum EN ENO In Out Member EndDat	Out:=RecNum(In, Member, EndDat);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array to process	Input	Array of structures to process	---	---	*
Member	Member to process		Member of *In[]* structure to process	Depends on data type.		
EndDat	End data		End data			
Out	Number of records	Output	Number of records	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)	Specify an array of structures.																			
Member	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
																*1				
	Enumerations can also be specified.*2																			
	Must be the same data type as the members to process in *In[]*.																			
EndDat	Must be the same data type as *Member*.																			
Out							OK													

*1 You can specify TIME, DATE, TOD, and DT data with CPU Units with unit version 1.01 or later and Sysmac Studio version 1.02 or higher.

*2 A CPU Unit with unit version 1.02 or later and Sysmac Studio version 1.03 or higher are required to specify enumerations.

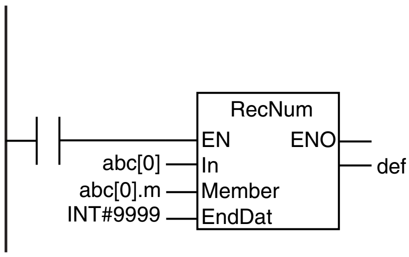
Function

The RecNum instruction searches from the start of an array *In[]* (whose elements are structures). The instruction searches for elements for which the value of member to process *Member* matches end data *EndDat*. As the result, it assigns the number of elements (records) up to the element just before the element with an *EndDat* match to *Out*. One of the members to process in the elements of *In[]* is passed as an argument to *Member*.

Always attach the element number to input parameter that is passed to *In[]*, e.g., *array[3]*.

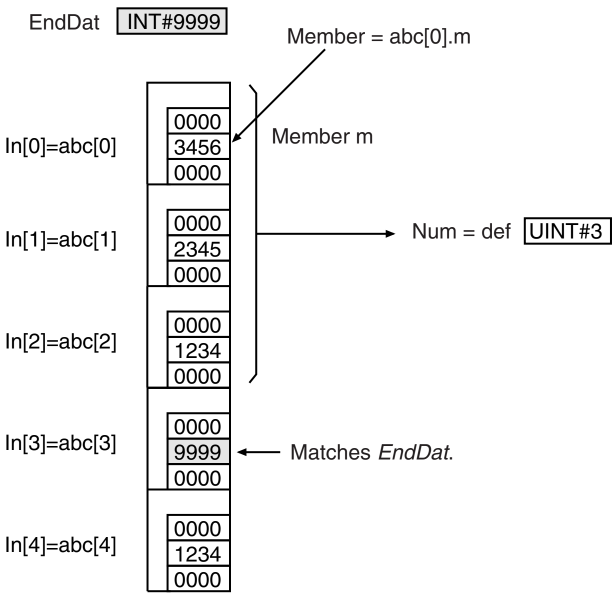
The following example is for when *EndDat* is INT#9999.

LD



ST

```
def:=RecNum(abc[0], abc[0].m, INT#9999);
```



Additional Information

- *In[]* can be a member of a higher-level structure.
  Example: In[0]=str0.str1[0]
- When you search TIME, DT, or TOD data, adjust the data so that the precision of *Member* and *EndDat* is the same. Use the following instructions to adjust the precision of the values: TruncTime (page 2-657), TruncDt (page 2-661), and TruncTod (page 2-665).

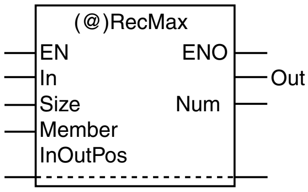
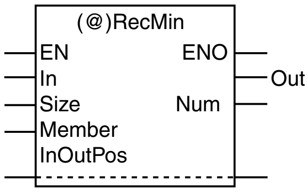
Precautions for Correct Use

- Use an array that is the element of a structure for *In[].* Otherwise, a building error will occur.
- The data types of *Member* and *EndDat* must be the same. If they are different, a building error will occur.
- If there are no members in *In[]* that match *EndDat*, the total number of elements in *In[]* is assigned to *Out*.
- If *Member* is a real number, depending on the value of *Member*, the desired results may not be achieved due to error.
- If *EndDat* is a real number, do not specify nonnumeric data for it.
- When an element in the array is passed to *In[]*, all elements below the passed element are processed.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - *Member* is not a member of *In[]*.
  - *Member* is STRING data and it does not end in a NULL character.

RecMax and RecMin

RecMax: Searches the specified member in the structures of an array of structures for the maximum value.

RecMin: Searches the specified member in the structures of an array of structures for the minimum value.

Instruction	Name	FB/FUN	Graphic expression	ST expression
RecMax	Maximum Record Search	FUN		Out:=RecMax(In, Size, Member, InOutPos, Num);
RecMin	Minimum Record Search	FUN		Out:=RecMin(In, Size, Member, InOutPos, Num);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array to search	Input	Array of structures to search	---	---	*
Size	Number of elements to search		Number of array elements to search	Depends on data type.		1
Member	Member to search		Member of I*n[]* structure to search			*
InOutPos[] (array)	Found element number	In-out	Found element number	Depends on data type.	---	---
Out	Search result	Output	Search result	Depends on data type.	---	---
Num	Number found		Number found			

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)	Specify an array of structures.																			
Size							OK													

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Member						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
																*				
	Specify the same data type as the search member of *In[]*.																			
InOutPos[] (array)							OK													
Out						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
																*				
Num							OK													

* You can specify TIME, DATE, TOD, DT, and STRING data with CPU Units with unit version 1.01 or later and Sysmac Studio version 1.02 or higher.

Function

These instructions search *Size* elements in an array of structures *In[]*. The search range is therefore from *In[0]* to *In[Size−1]*. The instruction searches member to search Member in the structures.

One of the members to search in the elements of *In[]* is passed as an argument to *Member*. The element number of the element with the maximum or minimum value is assigned to *InOutPos[0]* and the number of elements that were found is assigned to *Num*. If more than one element was found, the element number of the lowest element with the maximum or minimum value in *In[]* is assigned to *InOutPos[0]*.

Always attach the element number to input parameter that is passed to *In[]*, e.g., *array[3]*.

The relationship between values with data types that are not integers or real numbers are determined as given in the following table.

Data type	Relationship
TIME	The numerically larger value is considered to be larger.
DATE, TOD, or DT	Later dates or times of day are considered to be larger.
STRING	The specifications are the same as for the LTascii, LEascii, GTascii, and GEascii instructions (page 2-104). Refer to the specified page for details.

● RecMax

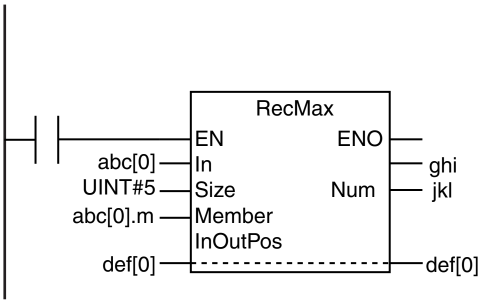
The RecMax instruction searches for the maximum value. The maximum value of the member to search is assigned to search result *Out.*

● RecMin

The RecMin instruction searches for the minimum value. The minimum value of the member to search is assigned to search result *Out.*

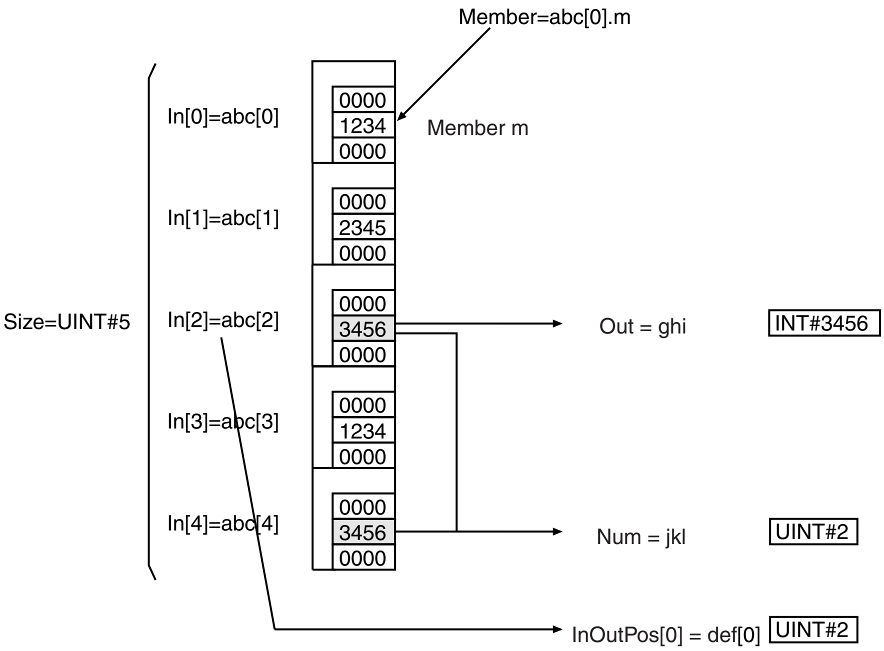
The following example shows the RecMax instruction when *Size* is UINT#5.

LD



ST

```
ghi:=RecMax(abc[0], UINT#5, abc[0].m, def[0], jkl);
```



Additional Information

- *In[]* can be a member of a higher-level structure.

  Example: In[0]=str0.str1[0]
- *In[]* can be an array with two or more dimensions. If *In[]* is a two-dimensional array, the element number in the first dimension of the element that matches the search conditions is assigned to *InOutPos[0]* and the element number in the second dimension is assigned to *InOutPos[1]*.
- If *In[]* is a three-dimensional array, the element number in the first dimension of the element that matches the search conditions is assigned to *InOutPos[0]*, the element number in the second dimension is assigned to *InOutPos[1]*, and the element number in the third dimension is assigned to *InOutPos[2]*.
- When you search TIME, DT, or TOD data, adjust the data so that the precision of *Member* is the same. Use the following instructions to adjust the precision of the values: TruncTime (page 2-657), TruncDt (page 2-661), and TruncTod (page 2-665).

Precautions for Correct Use

- If you use a different data type for *Member* and *Out*, use only the following data types and make sure the valid range of *Out* includes the valid range of *Member*.
  - USINT, UINT, UDINT, ULINT, SINT, INT, DINT, LINT, REAL, and LREAL
- If *Member* is a real number, depending on the value of *Member*, the desired results may not be achieved due to error.

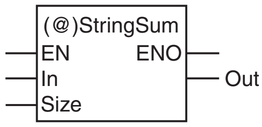
- When *In* is an enumeration, always use a variable for the input parameter to pass to *In*. A building error will occur if a constant is passed.
- If the value of *Size* is 0, the values of *Out* and Num are 0. If *Member* is STRING data and the value of *Size* is 0, *Out* is a text string containing only the NULL character. The values in *InOutPos[]* do not change.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out*, *InOutPos[]*, and *Num* will not change.
  - The value of *Size* exceeds the array area of *In[]*.
  - *Member* is not a member of *In[]*.
  - The array size of *InOutPos[]* is smaller than the number of dimensions of *In[]*.
  - *Member* is STRING data and it does not end in a NULL character.

FCS Instructions

Instruction	Name	Page
StringSum	Checksum Calculation	2-538
StringLRC	Calculate Text String LRC	2-540
StringCRCCCITT	Calculate Text String CRC-CCITT	2-542
StringCRC16	Calculate Text String CRC-16	2-544
AryLRC_**	Calculate Array LRC Group	2-546
AryCRCCCITT	Calculate Array CRC-CCITT	2-548
AryCRC16	Calculate Array CRC-16	2-550

StringSum

The StringSum instruction calculates the checksum for a text string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
StringSum	Checksum Calculation	FUN		Out:=StringSum(In, Size);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Text string to process	Input	Text string to process	Depends on data type.	---	''
Size	Byte size		Byte size of checksum	1 or 2	Bytes	1
Out	Checksum	Output	Checksum	Number of bytes specified by *Size*	Bytes	---

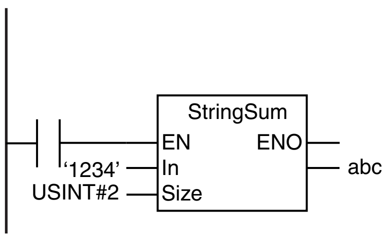
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
Size						OK														
Out																				OK

Function

The StringSum instruction calculates the checksum of text string to process *In*. Checksum *Out* will be the number of bytes specified with byte size *Size*. *Out* is given as a hexadecimal text string with a NULL character stored at the end.

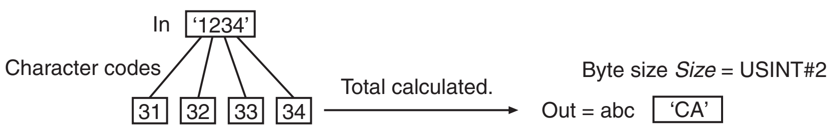
The following example is for when *In* is ‘1234’ and *Size* is USINT#2.

LD



ST

abc:=StringSum(’1234’, USINT#2);



If *Size* was USINT#1 in the above example, *Out* would be ‘A’.

Precautions for Correct Use

- If the sum of the character codes in *In* exceeds the number of digits of *Size*, the upper digits are discarded.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of *Size* is outside of the valid range.
  - The number of bytes in *In* is 0 (i.e., the NULL character only).

StringLRC

The StringLRC instruction calculates the LRC value (horizontal parity).

Instruction	Name	FB/FUN	Graphic expression	ST expression
StringLRC	Calculate Text String LRC	FUN	(@)StringLRC EN ENO In Out	Out:=StringLRC(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Text string to process	Input	Text string to process	Depends on data type.	---	''
Out	LRC value	Output	LRC value	3 bytes max. (two single-byte alphanumeric characters plus the final NULL character)	---	---

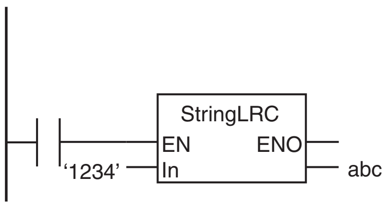
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
Out																				OK

Function

The StringLRC instruction calculates the LRC value (horizontal parity) of text string to process *In*. The LRC value is the exclusive logical OR of the character codes for the text string in *In.* The LRC value (*Out*) is given as a hexadecimal text string with a NULL character stored at the end.

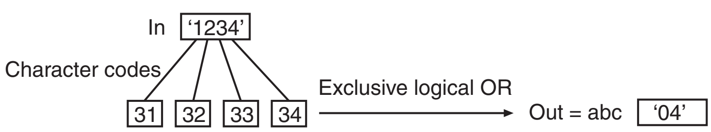
The following example is for when *In* is ‘1234’.

LD



ST

abc:=StringLRC(‘1234’);



Precautions for Correct Use

An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.

- The number of bytes in *In* is 0 (i.e., the NULL character only).

StringCRCCCITT

The StringCRCCCITT instruction calculates the CRC-CCITT value using the XMODEM method.

Instruction	Name	FB/FUN	Graphic expression	ST expression
StringCRCCCITT	Calculate Text String CRC-CCITT	FUN	(@)StringCRCCCITT EN ENO In Out Initial OutOrder	Out:=StringCRCCCITT(In, Initial, OutOrder);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Text string to process	Input	Text string to process	Depends on data type.	---	''
Initial	Initial value		Initial value of CRC-CCITT value			0
OutOrder	Byte order		Order to process bytes in *In*	_LOW_HIGH, _HIGH_LOW		_HIGH _LOW
Out	CRC-CCITT value	Output	CRC-CCITT value	5 bytes (four single-byte alphanumeric characters plus the final NULL character)	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
Initial			OK																	
OutOrder	Refer to *Function* for the enumerators of the enumerated type _eBYTE_ORDER.																			
Out																				OK

Function

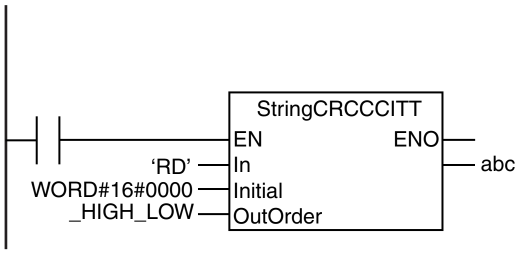
The StringCRCCCITT instruction calculates the CRC-CCITT value of text string to process *In* using the XMODEM method. CRC-CCITT value *Out* is given as a hexadecimal text string with a NULL character stored at the end.

Set *Initial* to the initial value for CRC-CCITT value calculation. *OutOrder* specifies the byte order.
The data type of *OutOrder* is enumerated type _eBYTE_ORDER. The meanings of the enumerators are as follows:

Enumerators	Meaning
_LOW_HIGH	Lower byte first, upper byte last
_HIGH_LOW	Upper byte first, lower byte last

The following example is for when *In* is ‘RD’, *Initial* is WORD#16#0000, and *OutOrder* is _HIGH_LOW.

LD



ST

```
abc:=StringCRCCCITT(’RD’, WORD#16#0000,
                    _HIGH_LOW);
```

Precautions for Correct Use

An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.

- The value of *OutOrder* is outside of the valid range.
- The number of bytes in *In* is 0 (i.e., the NULL character only).

StringCRC16

The StringCRC16 instruction calculates the CRC-16 value using the MODBUS method.

Instruction	Name	FB/FUN	Graphic expression	ST expression
StringCRC16	Calculate Text String CRC-16	FUN	(@)StringCRC16 EN ENO In Out Initial OutOrder	Out:=StringCRC16(In, Initial, OutOrder);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Text string to process	Input	Text string to process	Depends on data type.	---	''
Initial	Initial value		Initial value of CRC-16 value			16#FFFF
OutOrder	Byte order		Order to process bytes in *In*	_LOW_HIGH, _HIGH_LOW		_LOW_HIGH
Out	CRC-16 value	Output	CRC-16 value	5 bytes (four single-byte alphanumeric characters plus the final NULL character)	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
Initial			OK																	
OutOrder	Refer to *Function* for the enumerators of the enumerated type _eBYTE_ORDER.																			
Out																				OK

Function

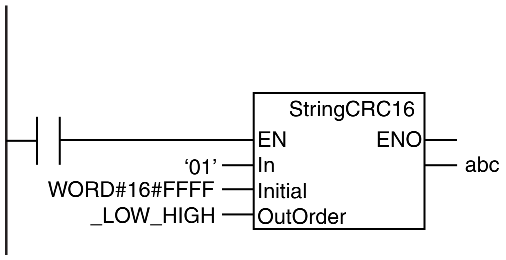
The StringCRC16 instruction calculates the CRC-16 value of text string to process *In* using the MODBUS method. CRC-16 value *Out* is given as a hexadecimal text string with a NULL character stored at the end.

Set *Initial* to the initial value for CRC-16 value calculation. *OutOrder* specifies the byte order.
The data type of *OutOrder* is enumerated type _eBYTE_ORDER. The meanings of the enumerators are as follows:

Enumerators	Meaning
_LOW_HIGH	Lower byte first, upper byte last
_HIGH_LOW	Upper byte first, lower byte last

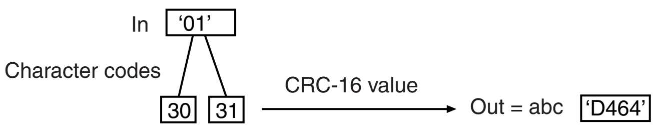
The following example is for when *In* is ‘01’, *Initial* is WORD#16#FFFF and *OutOrder* is _LOW_HIGH.

LD



ST

```
abc:=StringCRC16('01', WORD#16#FFF
                 _LOW_HIGH);
```



Precautions for Correct Use

An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.

- The value of *OutOrder* is outside of the valid range.
- The number of bytes in *In* is 0 (i.e., the NULL character only).

AryLRC_**

The AryLRC_** instructions calculate the LRC value for an array.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryLRC_**	Calculate Array LRC Group	FUN	(@)AryLRC_** EN ENO In Out Size "**" must be a bit string data type.	Out:=AryLRC_**(In, Size); "**" must be a bit string data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array to process	Input	Array to process	Depends on data type.	---	*
Size	Number of elements to process		Number of *In[]* elements			1
Out	LRC value	Output	LRC value	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)		OK	OK	OK	OK															
Size							OK													
Out	Must be same data type as *In[]*																			

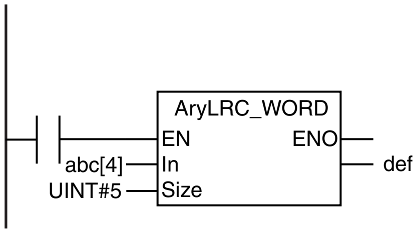
Function

The AryLRC_** instructions calculate the LRC value (exclusive logical OR) of *Size* array elements of array to process *In[]* starting from *In[0]*. The name of the instruction is determined by the data type of *In[]*. For example, if *In[]* is the WORD data type, the instruction is AryLRC_WORD.

Always attach the element number to in-out parameter that is passed to *In[]*, e.g., *array[3]*.

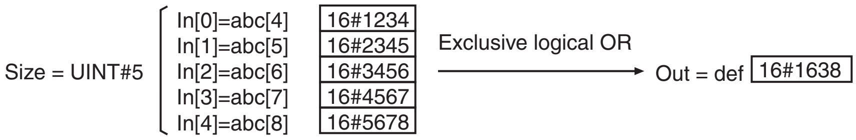
The following example shows the AryLRC_WORD instruction when *Size* is UINT#5.

LD



ST

```
def:=AryLRC_WORD(abc[4], UINT#5);
```



Precautions for Correct Use

- Use the same data type for *In[]* and *Out.*
- If the value of *Size* is 0, the value of *Out* is 16#00.
- An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.
  - The value of *Size* exceeds the array area of *In[]*.

AryCRCCCITT

The AryCRCCCITT instruction calculates the CRC-CCITT value using the XMODEM method.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryCRCCCITT	Calculate Array CRC-CCITT	FUN	(@)AryCRCCCITT EN ENO In Out Size Initial OutOrder	Out:=AryCRCCCITT(In, Size, Initial, OutOrder);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array to process	Input	Array to process	Depends on data type.	---	*
Size	Number of elements to process		Number of *In[]* elements			1
Initial	Initial value		Initial value of CRC-CCITT value			0
OutOrder	Byte order		Order to process bytes in *In*	_LOW_HIGH, _HIGH_LOW		_HIGH _LOW
Out	CRC-CCITT value	Output	CRC-CCITT value	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)		OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	
Size							OK													
Initial			OK																	
OutOrder	Refer to *Function* for the enumerators for the enumerated type _eBYTE_ORDER.																			
Out			OK																	

Function

The AryCRCCCITT instruction calculates the CRC-CCITT value of *Size* elements of array to process *In[]* starting from *In[0]*. The XMODEM method is used.

Set *Initial* to the initial value for CRC-CCITT value calculation. *OutOrder* specifies the byte order.

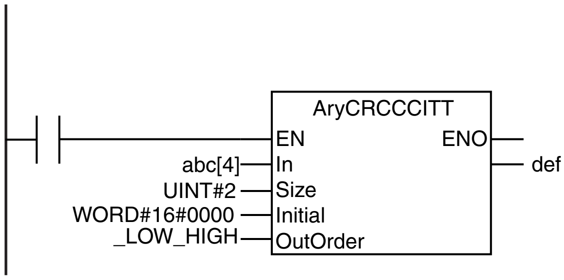
The data type of *OutOrder* is enumerated type _eBYTE_ORDER. The meaning of the enumerators are as follows:

Enumerators	Meaning
_LOW_HIGH	Lower byte first, upper byte last
_HIGH_LOW	Upper byte first, lower byte last

Always attach the element number to in-out parameter that is passed to *In[]*, e.g., *array[3]*.

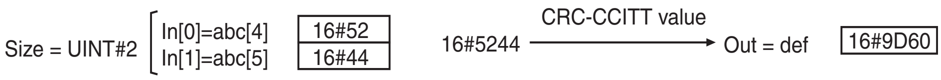
The following example is for when *Size* is UINT#2, *Initial* is WORD#16#0000, and *OutOrder* is _LOW_HIGH.

LD



ST

```
def:=AryCRCCCITT(abc[4], UINT#2,
                 WORD#16#0000,
                 _LOW_HIGH);
```



Precautions for Correct Use

- If the value of *Size* is 0, the value of *Out* is WORD#16#00.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of *OutOrder* is outside of the valid range.
  - The value of *Size* exceeds the array area of *In[]*.

AryCRC16

The AryCRC16 instruction calculates the CRC-16 value using the MODBUS method.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AryCRC16	Calculate Array CRC-16	FUN	(@)AryCRC16 EN → ENO In → Out Size Initial OutOrder	Out:=AryCRC16(In, Size, Initial, OutOrder);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In[] (array)	Array to process	Input	Array to process	Depends on data type.	---	*
Size	Number of elements to process		Number of *In[]* elements			1
Initial	Initial value		Initial value of CRC-16 value			16#FFFF
OutOrder	Byte order		Order to process bytes in *In*	_LOW_HIGH, _HIGH_LOW		_LOW _HIGH
Out	CRC-16 value	Output	CRC-16 value	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[] (array)		OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	
Size							OK													
Initial			OK																	
OutOrder	Refer to *Function* for the enumerators for the enumerated type _eBYTE_ORDER.																			
Out			OK																	

Function

The AryCRC16 instruction calculates the CRC-16 value of *Size* array elements of array to process *In[]* starting from *In[0]*. The MODBUS method is used.

Set *Initial* to the initial value for CRC-16 value calculation. *OutOrder* specifies the byte order.

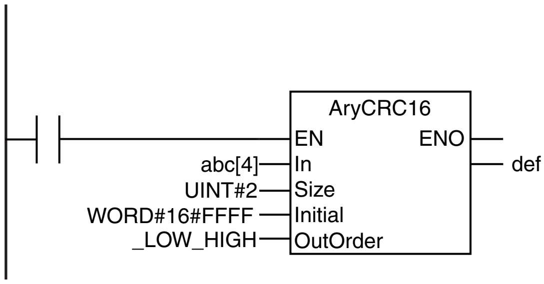
The data type of *OutOrder* is enumerated type _eBYTE_ORDER. The meaning of the enumerators are as follows:

Enumerator	Meaning
_LOW_HIGH	Lower byte first, upper byte last
_HIGH_LOW	Upper byte first, lower byte last

Always attach the element numbers to the input parameter that is passed to *In[]*, e.g., *array[3].*

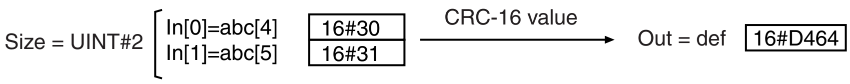
The following example is for when *Size* is UINT#2, *Initial* is WORD#16#FFFF and *OutOrder* is _LOW_HIGH.

LD



ST

```
def:=AryCRC16(abc[4], UINT#2,
              WORD#16#FFFF,
              _LOW_HIGH);
```



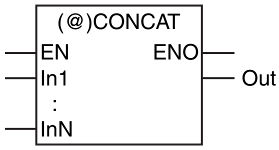
Precautions for Correct Use

- If the value of *Size* is 0, the value of *Out* is WORD#16#0.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of *OutOrder* is outside of the valid range.
  - The value of *Size* exceeds the array area of *In[]*.

Text String Instructions

CONCAT

The CONCAT instruction joins two to five text strings.

Instruction	Name	FB/FUN	Graphic expression	ST expression
CONCAT	Concatenate String	FUN		Out:=CONCAT(In1,···, InN);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1 to InN	Strings to join	Input	Text strings to join, where N is 2 to 5	Depends on data type.	---	''*
Out	Result of joining	Output	Text string that resulted from joining	Depends on data type.	---	---

* If you omit the input parameter that connects to *InN*, the default value is not applied, and a building error will occur. For example, if N is 3 and the input parameters that connect to *In1* and *In2* are omitted, the default values are applied, but if the input parameter that connects to *In3* is omitted, a building error will occur.

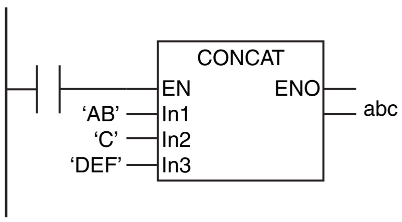
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN																				OK
Out																				OK

Function

The CONCAT instruction joins 2 to 5 text strings in strings to join *In1* to *InN* in that order. It adds a NULL character to the end.

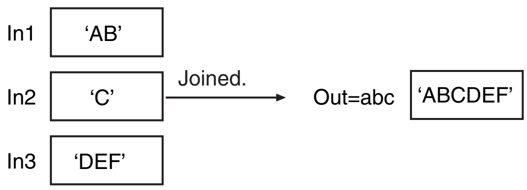
The following example is for when *In1* is ‘AB’, *In2* is ‘C’ and *In3* is ‘DEF’. The value of variable *abc* will be ‘ABCDEF’.

LD



ST

abc:=CONCAT(’AB’, ‘C’, ‘DEF’);



Precautions for Correct Use

An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.

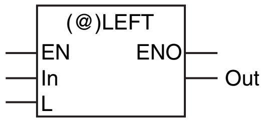
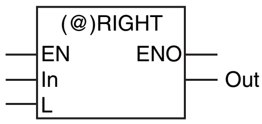
- The length of the joined character strings exceeds 1,986 bytes.

LEFT and RIGHT

These instructions extract a text string with the specified number of characters.

LEFT: Extracts characters from the left (beginning) of the text string.

RIGHT: Extracts characters from the right (end) of the text string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
LEFT	Get String Left	FUN		Out:=LEFT(In, L);
RIGHT	Get String Right	FUN		Out:=RIGHT(In, L);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Source string	Input	Text string from which to extract characters	Depends on data type.	---	''
L	Number of characters		Number of characters to extract	0 to 1985		1
Out	Extraction result	Output	Extracted text string	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
L							OK													
Out																				OK

Function

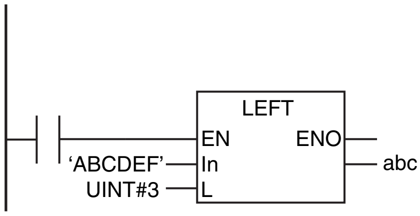
These instructions extract a text string with the number of characters specified by number of characters *L* from the source string *In*. A NULL character is placed at the end of extraction result *Out.*

● LEFT

Extracts characters from the left (beginning) of *In*.

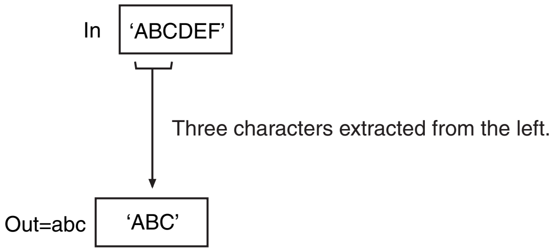
The following example is for when *In* is ‘ABCDEF’ and *L* is UINT#3. The value of variable *abc* will be ‘ABC’.

LD



ST

```
abc:=LEFT(‘ABCDEF’, UINT#3);
```

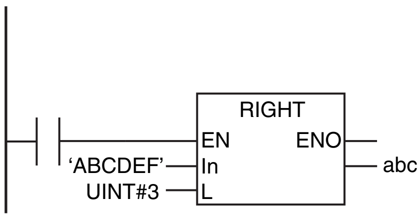


RIGHT

Extracts characters from the right (end) of *In*.

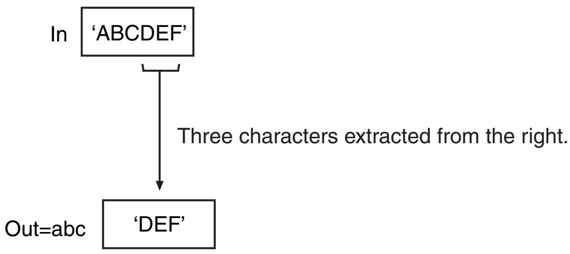
The following example is for when *In* is ‘ABCDEF’ and *L* is UINT#3. The value of variable *abc* will be ‘DEF’.

LD



ST

```
abc:=RIGHT(‘ABCDEF’, UINT#3);
```

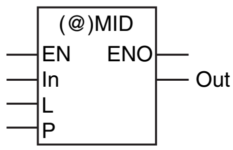


Precautions for Correct Use

- If the value of *L* is larger than the number of characters in *In* or it is within the valid range, an error does not occur and all of the characters in *In* are copied to *Out*.
- If the value of *L* is 0, an error does not occur and only the NULL character is assigned to *Out*.
- Multi-byte characters are counted as one character each.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - *In* results in a character code error.

MID

The MID instruction extracts a text string with the specified number of characters from the specified character position.

<table>
<tr><th>Instruction</th><th>Name</th><th>FB/FUN</th><th>Graphic expression</th><th>ST expression</th></tr>
<tr><td>MID</td><td>Get String Any</td><td>FUN</td><td>
</td><td>Out:=MID(In, L, P);</td></tr>
</table>

Variables

<table>
<tr><th>Name</th><th>Meaning</th><th>I/O</th><th>Description</th><th>Valid range</th><th>Unit</th><th>Default</th></tr>
<tr><td>In</td><td>Source string</td><td rowspan="3">Input</td><td>Text string from which to extract characters</td><td>Depends on data type.</td><td rowspan="3">---</td><td>''</td></tr>
<tr><td>L</td><td>Number of characters</td><td>Number of characters to extract</td><td>0 to 1985</td><td rowspan="2">1</td></tr>
<tr><td>P</td><td>First character</td><td>First character to extract</td><td>1 to 1985</td></tr>
<tr><td>Out</td><td>Extraction result</td><td>Output</td><td>Extracted text string</td><td>Depends on data type.</td><td>---</td><td>---</td></tr>
</table>

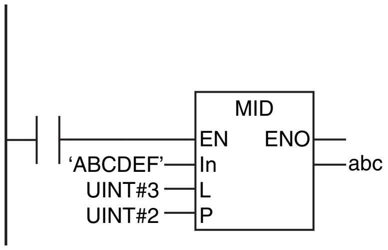
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
L							OK													
P							OK													
Out																				OK

Function

The MID instruction extracts a text string with the number of characters specified by number of characters *L* from the source string *In*. The first character to extract is specified by first character *P*. A NULL character is placed at the end of extraction result *Out.*

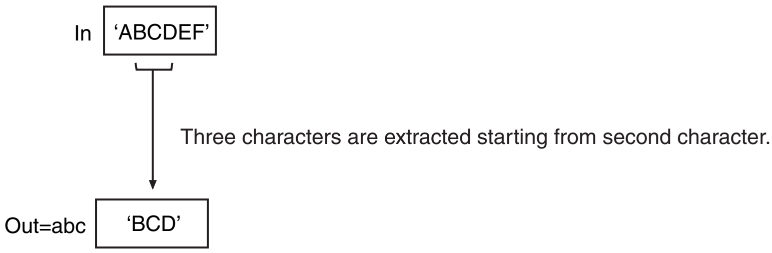
The following example is for when *In* is ‘ABCDEF’, *L* is UINT#3, and *P* is UINT#2. The value of variable *abc* will be ‘BCD’.

LD



ST

```
abc:=MID(‘ABCDEF’, UINT#3, UINT#2);
```

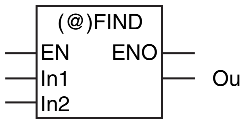


Precautions for Correct Use

- If the value of *L* is 0, an error does not occur and only the NULL character is assigned to *Out*.
- Multi-byte characters are counted as one character each.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - *In* results in a character code error.
  - *In* does not have enough characters for the number of characters specified by *L* from the position specified by *P*.
  - The value of *P* is 0.

FIND

The FIND instruction searches a specified text string for the position of a specified text string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FIND	Find String	FUN		Out:=FIND(In1, In2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	String to search	Input	Text string to search	Depends on data type.	---	''
In2	Search key		Text string to search for			
Out	Search result	Output	Search result	0 to 1985	---	---

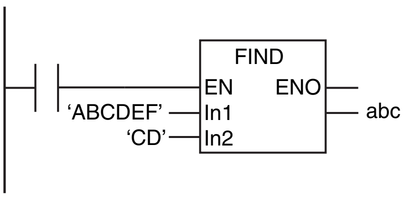
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																				OK
In2																				OK
Out							OK													

Function

The FIND instruction searches for search key *In2* in string to search *In1*. The position of *In2* from the start of *In1* is assigned to search result *Out*. If *In2* is not found in *In1*, *Out* is 0.

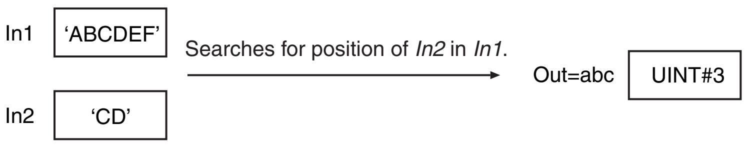
The following example is for when *In1* is 'ABCDEF' and *In2* is 'CD'. The value of variable *abc* will be UINT#3.

LD



ST

```
abc:=FIND('ABCDEF', 'CD');
```

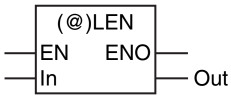


Precautions for Correct Use

- Make sure the number of characters in *In2* is less than the number of characters in *In1*. Otherwise, the value of *Out* will be 0.
- If *In2* exists more than once in *In1*, the position of the first *In2* from the beginning of *In1* is assigned to *Out*.
- If the value of *In1* and *In2* is only the NULL character, the value of *Out* is 1.
- Multi-byte characters are counted as one character each.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - *In1* or *In2* results in a character code error.

LEN

The LEN instruction finds the number of characters in a text string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
LEN	String Length	FUN		Out:=LEN(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Length string	Input	Text string to find length	Depends on data type.	---	''
Out	Find result	Output	Length detection result	0 to 1985	---	---

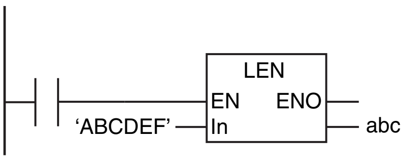
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
Out							OK													

Function

The LEN instruction finds the number of characters in length string *In*. A NULL character at the end of *In* is not counted.

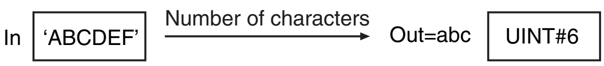
The following example is for when *In* is ‘ABCDEF’. The value of variable *abc* will be UINT#6.

LD



ST

abc:=LEN(‘ABCDEF’);

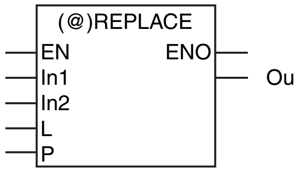


Precautions for Correct Use

- Multi-byte characters are counted as one character each.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - *In* results in a character code error.

REPLACE

The REPLACE instruction replaces part of a text string with another text string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
REPLACE	Replace String	FUN		Out:=REPLACE(In1, In2, L, P);

Variables

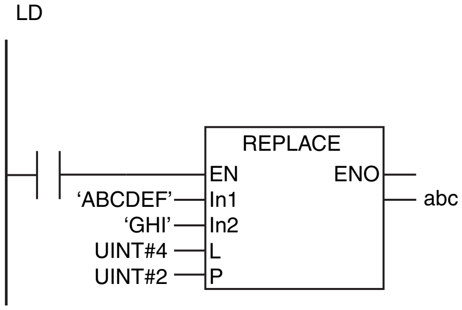
Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	String for replacement	Input	Text string for replacement	Depends on data type.	---	''
In2	Insert string		Text string to insert			
L	Number of characters		Number of characters to delete	0 to 1985		1
P	Replacement start position		Replacement start position	1 to 1985		
Out	Replacement result	Output	Text string after replacement	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																				OK
In2																				OK
L							OK													
P							OK													
Out																				OK

Function

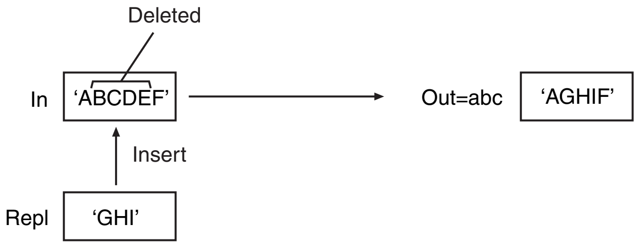
The REPLACE instruction replaces part of string for replacement *In1* with string to insert *In2*. First the number of characters specified by *L* from the position specified by *P* are deleted from *In1*. *In2* is then inserted for the deleted characters. A NULL character is placed at the end of replacement result *Out.*

The following example is for when *In1* is ‘ABCDEF’, *In2* is ‘GHI’, *P* is UINT#2, and *L* is UINT#4. The value of variable *abc* will be ‘AGHIF’.



ST

```
abc:=REPLACE(‘ABCDEF’, ‘GHI’, UINT#4, UINT#2);
```

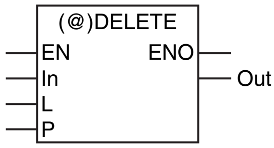


Precautions for Correct Use

- If *L* is 0, an error will not occur and all of the characters in *In1* are inserted to *Out*.
- If the value of *In2* is 0, *L* characters are deleted from *P* in *In1*.
- Multi-byte characters are counted as one character each.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - *In1* results in a character code error.
  - *In1* does not have enough characters for the number of characters specified by *L* from the position specified by *P*.
  - The value of *P* is 0.
  - The length of the character string after the replacement exceeds 1,986 bytes.

DELETE

The DELETE instruction deletes all or part of a text string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
DELETE	Delete String	FUN		Out:=DELETE(In, L, P);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	String for deletion	Input	Text string for deletion	Depends on data type.	---	''
L	Number of characters		Number of characters to delete	0 to 1985		1
P	Deletion start position		Deletion start position	1 to 1985		
Out	Deletion result	Output	Text string after deletion	Depends on data type.	---	---

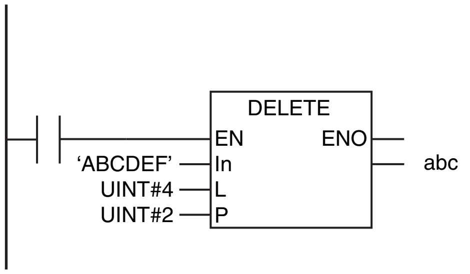
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
L							OK													
P							OK													
Out																				OK

Function

The DELETE string deletes the number of characters specified by *L* from the position specified by *P* from *In*. A NULL character is placed at the end of deletion result *Out.*

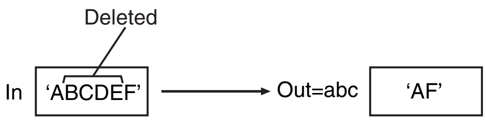
The following example is for when *In* is 'ABCDEF', *L* is UINT#4, and *P* is UINT#2. The value of variable *abc* will be 'AF'.

LD



ST

```
abc:=DELETE('ABCDEF', UINT#4, UINT#2);
```

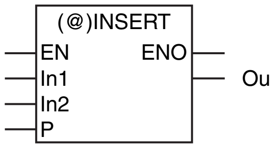


Precautions for Correct Use

- If *L* is 0, an error will not occur and all of the characters in *In* are inserted to *Out*.
- Multi-byte characters are counted as one character each.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - *In* results in a character code error.
  - *In* does not have enough characters for the number of characters specified by *L* from the position specified by *P*.
  - The value of *P* is 0.

INSERT

The INSERT instruction inserts a text string into another text string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
INSERT	Insert String	FUN		Out:=INSERT(In1, In2, P);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Original string	Input	Text string into which to insert string	Depends on data type.	---	''
In2	Insert string		Text string to insert			
P	Insertion start position		Insertion start position	0 to 1985		0
Out	Insertion result	Output	Text string after insertion	Depends on data type.	---	---

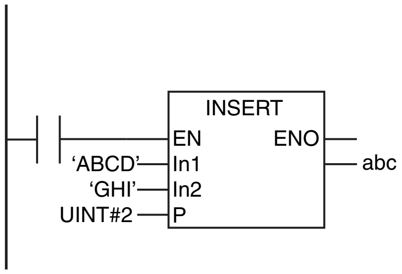
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																				OK
In2																				OK
P							OK													
Out																				OK

Function

The INSERT instruction inserts insertion string *In2* into original string *In1* at insertion start position *P*. A NULL character is placed at the end of insertion result *Out.*

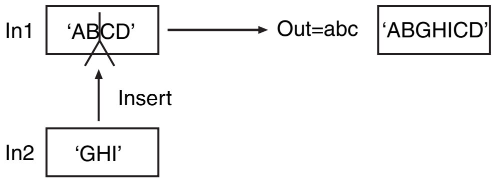
The following example is for when *In1* is 'ABCD', *In2* is 'GHI', and *P* is UINT#2. The value of variable *abc* will be 'ABGHICD'.

LD



ST

```
abc:=INSERT('ABCD', 'GHI', UINT#2);
```



Additional Information

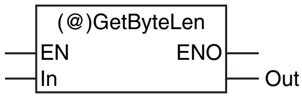
If *P* is 0, *In1* is inserted at the end of *In2*.

Precautions for Correct Use

- Multi-byte characters are counted as one character each.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - *In1* results in a character code error.
  - The value of *P* is greater than the number of characters in *In1*.
  - The length of the character string after the insertion exceeds 1,986 bytes.

GetByteLen

The GetByteLen instruction counts the number of bytes in a text string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
GetByteLen	Get Byte Length	FUN		Out:=GetByteLen(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Count string	Input	Text string to count number of bytes	Depends on data type.	---	''
Out	Number of bytes	Output	Number of bytes	0 to 1985	Bytes	---

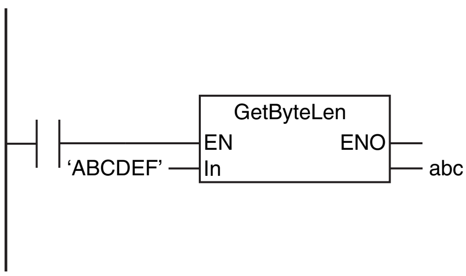
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
Out							OK													

Function

The GetByteLen instruction counts the number of bytes in count string *In*. A NULL character at the end of the text string is not counted.

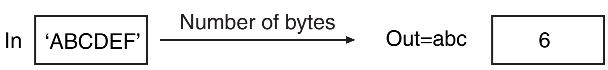
The following example is for when *In* is ‘ABCDEF’. The value of variable *abc* will be 6.

LD



ST

abc:=GetByteLen(‘ABCDEF’);

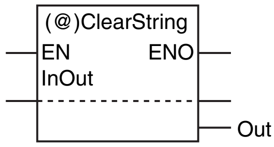


Additional Information

If *In* contains only ASCII characters, the result will be the same as the result of the LEN instruction.

ClearString

The ClearString instruction clears a text string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ClearString	Clear String	FUN		ClearString(InOut);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
InOut	Clear string	In-out	Text string to clear	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

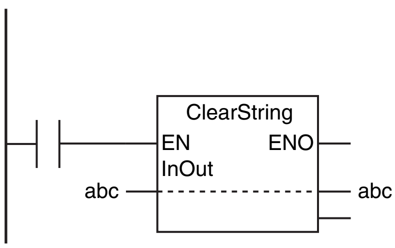
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
InOut																				OK
Out	OK																			

Function

The ClearString instruction clears clear string *InOut*. NULL characters are stored in the entire range of *InOut*.

The following figure shows a programming example. The content of STRING variable will be all NULL characters.

LD



ST

```
ClearString(abc);
```

The ClearString instruction stores NULL characters in the entire range of *InOut*.

The following example is for when *abc* is a 5-character STRING variable.

InOut=abc					
	NULL	NULL	NULL	NULL	NULL

Precautions for Correct Use

Return value *Out* is not used when the instruction is used in ST.

ToUCase and ToLCase

ToUCase: Converts all single-byte letters in a text string to uppercase.

ToLCase: Converts all single-byte letters in a text string to lowercase.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ToUCase	Convert to Uppercase	FUN	(@)ToUCase EN ENO In Out	Out:=ToUCase(In);
ToLCase	Convert to Lowercase	FUN	(@)ToLCase EN ENO In Out	Out:=ToLCase(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Data to convert	Input	Text string to convert	Depends on data type.	---	''
Out	Conversion result	Output	Converted text string	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
Out																				OK

Function

● ToUCase

The ToUCase instruction converts all single-byte letters in data to convert *In* to uppercase.

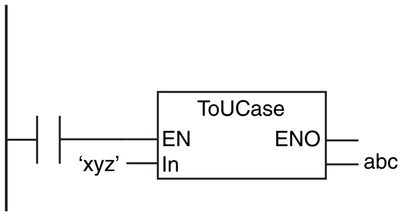
● ToLCase

The ToLCase instruction converts all single-byte letters in data to convert *In* to lowercase.

Both instructions output a NULL character at the end of the text string. Only single-byte characters are changed.

The following example for the ToUCase instruction is for when *In* is ‘xyz’. The value of variable *abc* will be ‘XYZ’.

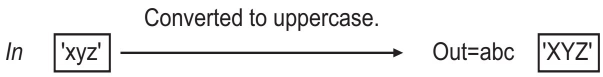
LD



ST

```
abc:=ToUCase(‘xyz’);
```

The ToUCase instruction converts all single-byte letters in *In* to uppercase.



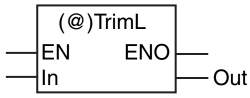
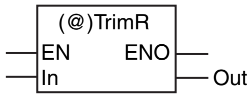
Precautions for Correct Use

- Two-byte letters are not converted.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - *In* results in a character code error.

TrimL and TrimR

TrimL: Removes blank space from the beginning of a text string.

TrimR: Removes blank space from the end of a text string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TrimL	Trim String Left	FUN		Out:=TrimL(In);
TrimR	Trim String Right	FUN		Out:=TrimR(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	String to trim	Input	Text string to trim	Depends on data type.	---	''
Out	Trimming result	Output	Text string after trimming	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
Out																				OK

Function

● TrimL

The TrimL instruction deletes blank characters from the beginning of string to trim *In*. If there are no blank characters at the beginning of the text string, nothing is done.

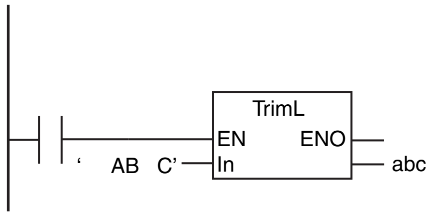
● TrimR

The TrimR instruction deletes blank characters from the end of string to trim *In*. If there are no blank characters at the end of the text string, nothing is done.

Both instructions output a NULL character at the end of the text string. Both ASCII spaces (16#20) and two-byte Japanese spaces (16#E38080) are treated as blank characters.

The following example for the TrimL instruction is for when *In* is '   AB   C'. The value of variable *abc* will be '   AB   C'.

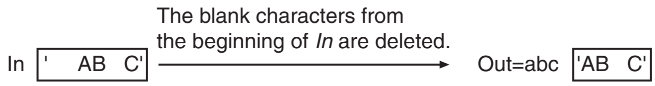
LD



ST

```
abc:=TrimL('    AB   C');
```

The TrimL instruction deletes blank characters from the beginning of *In*.



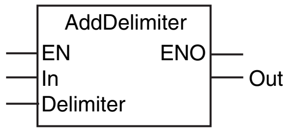
Precautions for Correct Use

An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.

- *In* results in a character code error.

AddDelimiter

The AddDelimiter instruction converts the values in a structure to text strings and adds delimiters.

Instruction	Name	FB/FUN	Graphic expression	ST expression
AddDelimiter	Put Text Strings with Delimiters	FUN		Out:=AddDelimiter(In,Delimiter);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Input structure	Input	Structure to convert to text strings	Depends on data type of members.	---	*
Delimiter	Delimiter		Delimiter	_COMMA _TAB _SEMICOLON _SPACE	---	_COMMA
Out	Return value	Output	Text strings with delimiters	1,986 bytes max. (1,985 single-byte alphanumeric characters plus the final NULL character)	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	Structure																			
Delimiter	Refer to *Function* for the enumerators for the enumerated data type _eDELIMITER.																			
Out																				OK

Function

The AddDelimiter instruction starts from the beginning of input structure *In* and converts the values of the members to text strings, which it separates with delimiter *Delimiter* and then concatenates. The concatenated text string is output to return value *Out*. A NULL character is placed at the end of *Out*.

The data type of *Delimiter* is enumerated type _eDELIMITER. The meanings of the enumerators are as follows:

Enumerator	Meaning
_COMMA	‘,’ (comma)
_TAB	‘$T’ (tab)
_SEMICOLON	‘;’(semicolon)
_SPACE	‘ ‘ (blank character)

The values of the members of *In* are converted according to their data type, as described next.

Boolean Data

FALSE is converted to ‘0’ and TRUE is converted to ‘1’.

Bit String Data

Bits strings are treated as hexadecimal numbers and converted to text strings that express them as alphanumeric characters. The 16# prefix of the hexadecimal number is not output to the text string.

If the value of the member requires fewer digits than are provided by the data type of the member, the upper digits will contain ‘0’. In other words, the unused digits are padded with zeros.

The number of characters in the text string depends on the data type as shown in the following table.

Data type of member	Number of characters
BYTE	2 single-byte alphanumeric characters
WORD	4 single-byte alphanumeric characters
DWORD	8 single-byte alphanumeric characters
LWORD	16 single-byte alphanumeric characters

Examples are given below.

Value of member	Converted text string
BYTE#16#AB	'AB'
LWORD#16#0123	'0000000000000123'

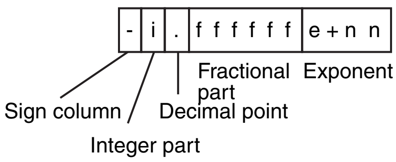
Integer Data

The value of the integer is converted to a text string. Upper digits that are 0 are not output to the text string. If the value of the member is negative, a minus sign (–) is added to the front of the text string. Examples are given below.

Value of member	Converted text string
UINT#0012	'12'
LINT#–12	'–12'

Real Number Data

The structure of the text string to which the value of the member is converted is shown below.



Item	Description
Sign column	If the value of the member is negative, a minus sign (–) is added. If the value of the member is positive, a plus sign (+) is not added.
Integer part	The integer part is always only one digit.
Decimal point	The decimal point is always given even if the value of the member is not a decimal number.
Fractional part	If the member is REAL data, 6 digits are given. If the member is LREAL data, 14 digits are given.
Exponent	The exponent is always given. ‘e’ indicates the exponent e. “nn” is 2 or 3 digits. The sign of “nn” is positive (+) if the absolute value of the member is 1.0 or higher and negative (–) if it is less than 1.0. If the value of the member is 0, this portion is ‘+’(positive).

If the value of the member is infinity, or nonnumeric data, the text string will be as shown below.

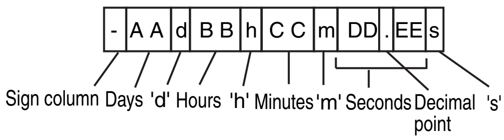
Value of member	Text string
+∞	'inf'
–∞	'–inf'
Nonnumeric data	‘nan’ or ‘–nan’

Examples are given below.

Value of member	Converted text string
REAL#3.14e1	'3.140000e+01'
REAL#–123.4567	'–1.234567e+02'
REAL#0	'0.000000e+00'
LREAL#0.00123456789	'1.23456789000000e–03'
LREAL#1.0e308	'1.00000000000000e+308'

Duration Data

The structure of the text string to which the value of the member is converted is shown below.



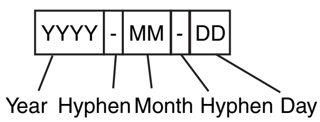
Item	Description
Sign column	If the value of the member is negative, a minus sign (–) is added. If the value of the member is positive, a plus sign (+) is not added.
Days	The number of days is always given. The range of the value is 0 to 106751. Upper digits are not padded with 0.
Hours	The number of hours is always given in two digits. The range of the value is 00 to 23.
Minutes	The number of minutes is always given in two digits. The range of the value is 00 to 59.
Seconds	The number of seconds is always given. The value of DD is always given in two digits between 00 and 59. The value of EE is always given in two digits between 000000000 and 999999999
‘d’, ‘h’, ‘m’, ‘s’, and the decimal point	These are always given.

Examples are given below.

Value of member	Converted text string
T#-180122000ms	'-2d02h02m02.000000000s'
T#100d2h3m5.678s	'100d02h03m05.678000000s'
T#2h3m5.678s	'0d02h03m05.678000000s'

Date Data

The structure of the text string to which the value of the member is converted is shown below.



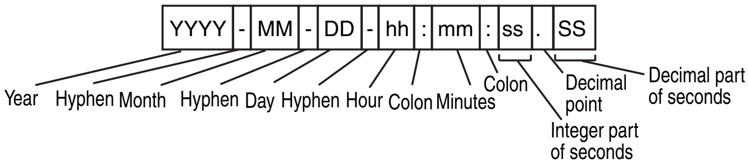
The month and day are converted to two digits each and output to the text string.

An example is shown below.

Value of member	Converted text string
D#2010-1-2	'2010-01-02'

Date and Time Data

The structure of the text string to which the value of the member is converted is shown below.



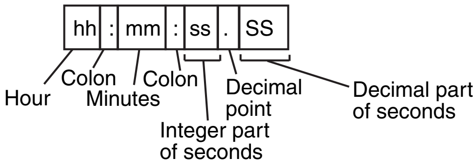
The month (MM), day (DD), hour (hh), minutes (mm), and integer part of the seconds (ss) are converted to two digits each and output to the text string. The fractional part of the seconds (ss) is converted to nine digits and output to the text string.

An example is shown below.

Value of member	Converted text string
DT#2004-09-23-12:16:8.12	'2004-09-23-12:16:08.120000000'

Time of Day Data

The structure of the text string to which the value of the member is converted is shown below.



The hour (hh), minutes (mm), and integer part of the seconds (ss) are converted to two digits each and output to the text string. The fractional part of the seconds (ss) is converted to nine digits and output to the text string.

An example is shown below.

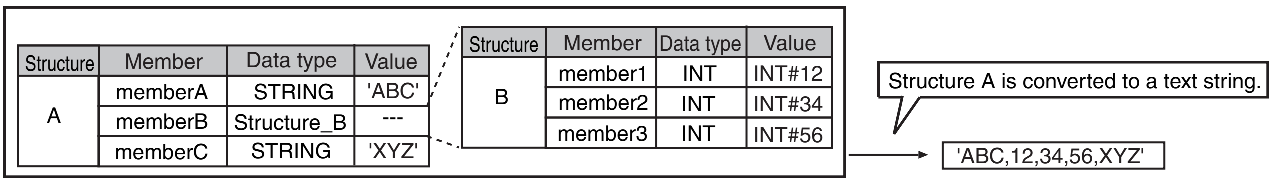
Value of member	Converted text string
TOD#2:16:28.12	'02:16:28.120000000'

Text String Data

The text string is output without any changes. The NULL character at the end of the text string is not included. For example, if the value of the member is ‘ABC’ and includes a NULL character at the end, ‘ABC’ without the NULL character is output to the text string.

Structure Data

The values of the members are converted in order from the start of the structure down to the nesting levels that are not structures. The values of the members are converted to text strings according to the rules for their data types. For example, if a member of structure A has a data type of Structure_B, the conversion works as shown below. Commas are used as delimiters in this example.



Enumeration Data

The value of the enumeration is treated as DINT data and converted accordingly.

For example, assume that an enumeration *Color* has three enumerators: *red*, *yellow*, and *green*. The numbers associated with these enumerators are as follows: *red* = 1, *yellow* = 2, *green* = 3. If the value of a member of enumeration *Color* is *yellow*, the text string will be ‘2’.

- **Array Data**

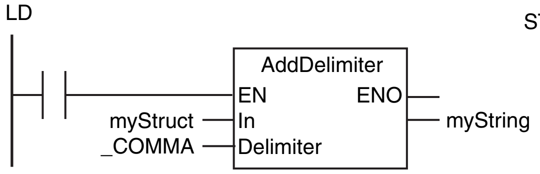
The text strings for the elements of the array are separated with the delimiter. The value of each element is converted according to the conversion rules for the data type of the array. Only one-dimensional arrays are converted.

For example, take the INT array *myArray[0..2]*. If the value of *myArray[0]* is INT#225, the value of *myArray[1]* is INT#-128, the value of *myArray[2]* is INT#0, and the delimiter is a comma, the text string would be as follows: ‘225,–128,0’.

Notation Example

The following notation is used to convert the myStruct structure to the myString text string. The ‘,’ (comma) is the delimiter.

LD



ST

```
myString:=AddDelimiter(myStruct,_COMMA);
```

Structure	Member	Data type	Value
myStruct	member1	STRING	'XYZ'
	member2	INT	INT#12
	member3	WORD	WORD#16#00AB
	member4	REAL	REAL#3.14e0

Converted to a text string.

Out = myString 'XYZ,12,00AB,3.140000e+00'

Additional Information

- You can combine this instruction with the *FilePuts instruction* (page 2-1301) to easily write values to specified CSV files in an SD Memory Card. Refer to *Sample Programming* for an application example.
- You can use the *SubDelimiter instruction* (page 2-588) to read text strings that were converted with the AddDelimiter instruction and output them as the values of the members of a structure.

Precautions for Correct Use

- Do not use the delimiter in the values of the members of *In*. If the delimiter is used in the values of the members of *In*, the SubDelimiter instruction will not correctly convert the text strings to the values of the members of a structure.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The text string that resulted from conversion exceeds 1,986 bytes, including the final NULL character.
  - A member of *In* is an array with more than one dimension.
  - A member of *In* is a union.

Version Information

A CPU Unit with unit version 1.02 or later and Sysmac Studio version 1.03 or higher are required to use this instruction.

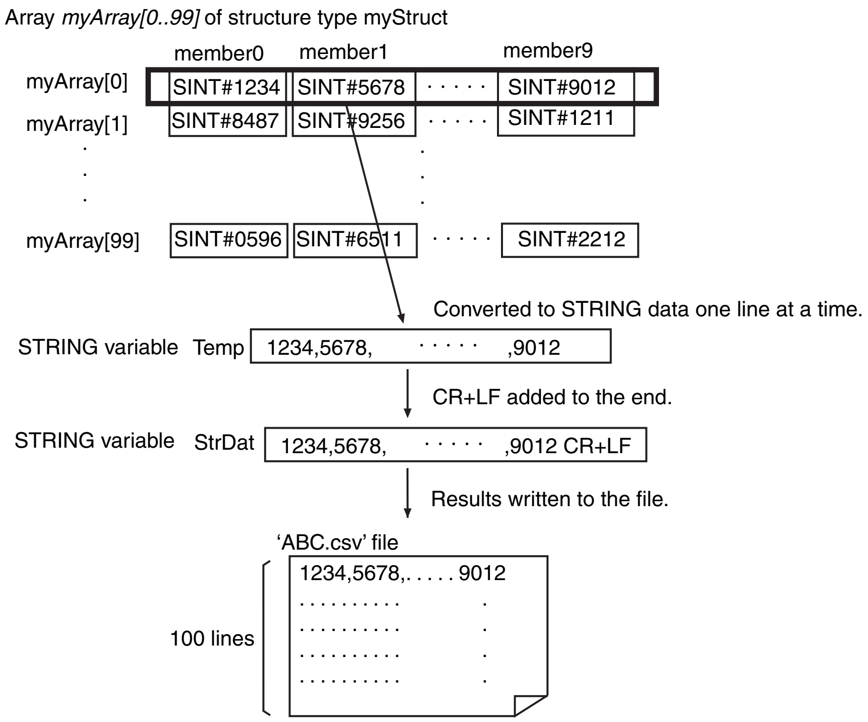
Sample Programming

The *myStruct* structure has ten members that are SINT variables. Here, the contents of *myArray[0..99]*, which is an array of structure type myStruct, are stored in 100 lines of a file named 'ABC.csv' in CSV file format in the SD Memory Card. Each line contains the values of the members of an array element converted to 10 text strings. Commas are inserted between them. A CR+LF code is added to the end of each line.

The processing procedure is as follows:

1. The FileOpen instruction is used to open the file 'ABC.csv.'
2. The AddDelimiter instruction is used to convert an element of *myArray[]* for one line and output the results to the *Temp* STRING variable.
3. The CONCAT instruction is used to concatenate *Temp* and CR+LF and then store the results in the *StrDat* STRING variable.
4. *StrDat* is written to the file.
5. Steps 2 to 4 are repeated for 100 lines.
6. The FileClose instruction is used to close the file.

Structure	Member	Data type
myStruct	member0	SINT
	member1	SINT
	⋮	
	member9	SINT



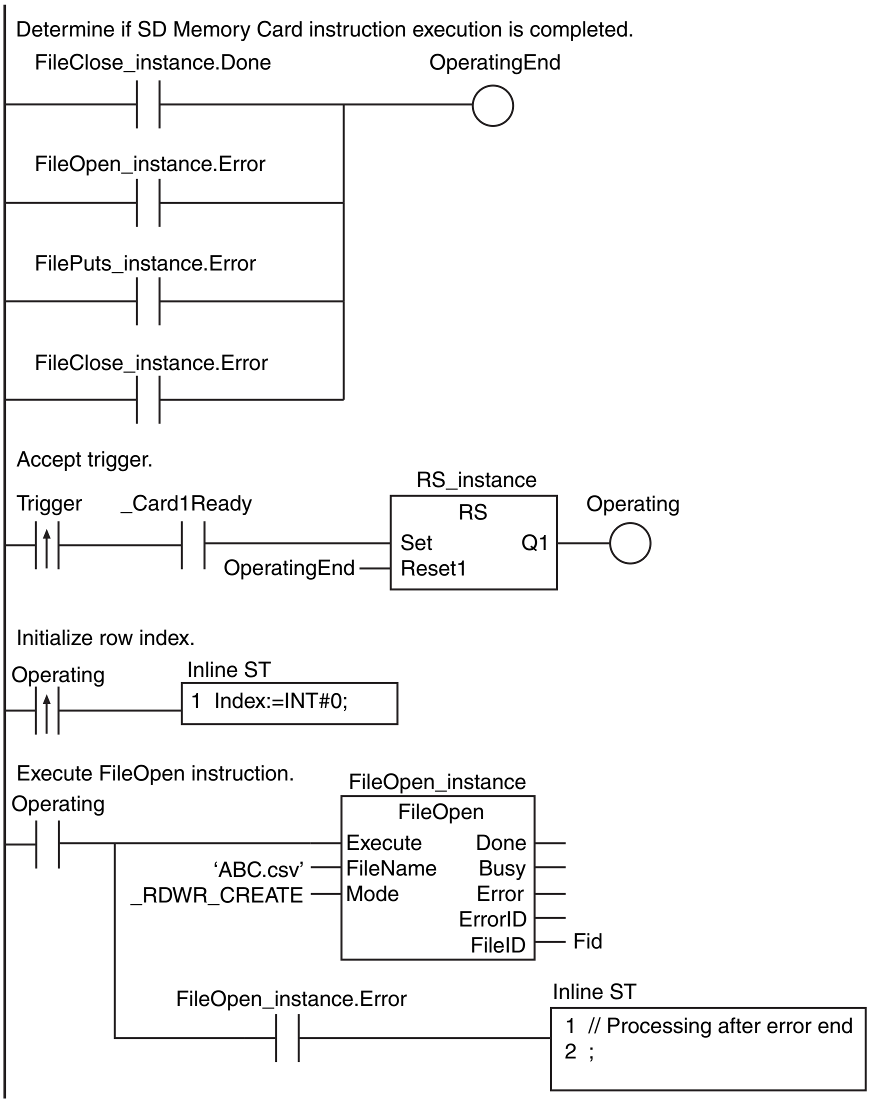
● Data Type Definition

Name	Data type	Comment
myStruct	STRUCT	Structure
member0	SINT	Member
member1	SINT	Member
member2	SINT	Member
member3	SINT	Member
member4	SINT	Member
member5	SINT	Member
member6	SINT	Member
member7	SINT	Member
member8	SINT	Member
member9	SINT	Member

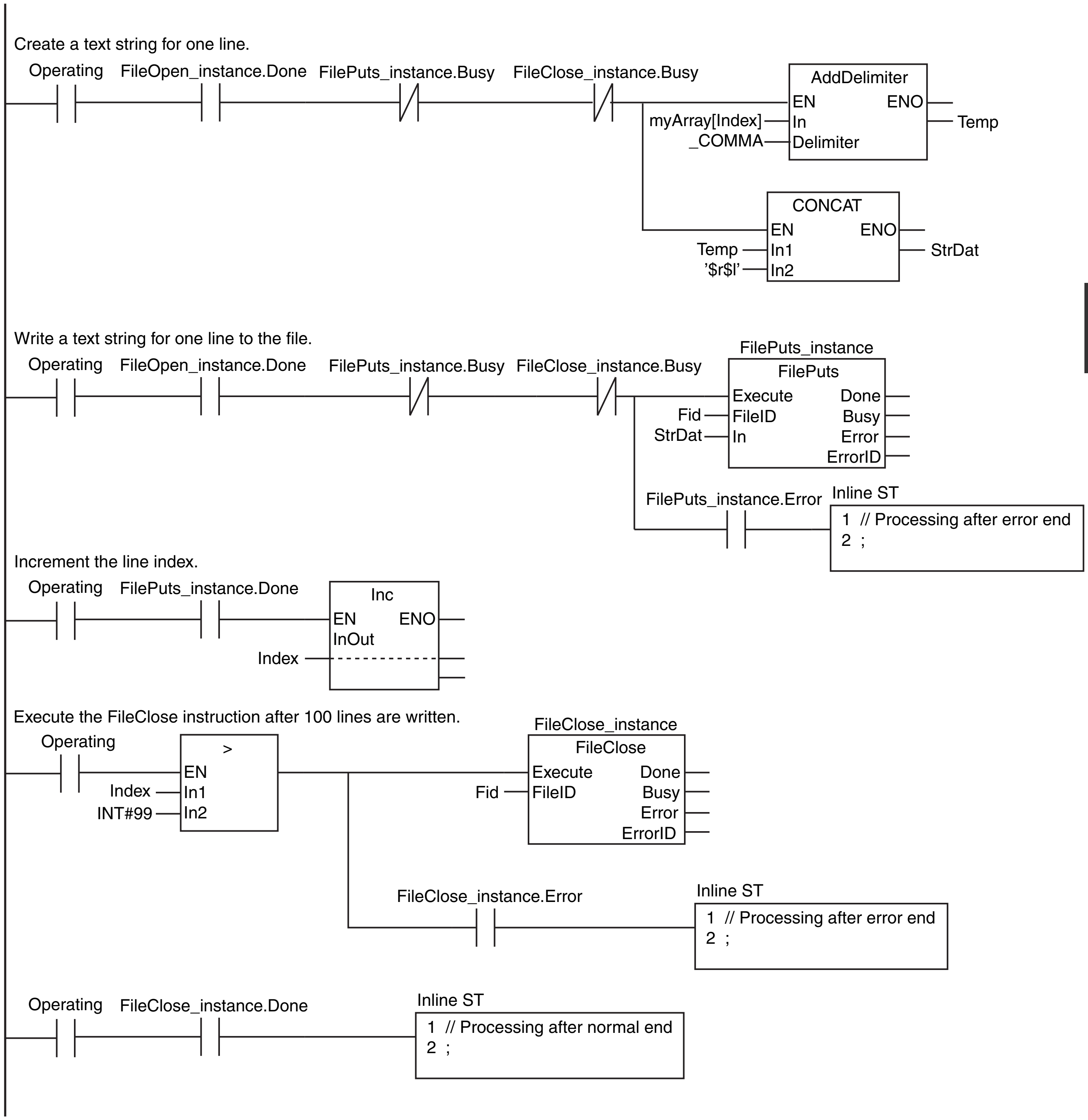
LD

Internal Variables	Variable	Data type	Initial value	Comment
	OperatingEnd	BOOL	False	Processing completed
	Trigger	BOOL	False	Execution condition
	Operating	BOOL	False	Processing
	Index	INT	0	Index
	Fid	DWORD	16#0	File ID
	StrDat	STRING[256]	''	Text string data
	myArray	ARRAY[0..99] OF myStruct	[100((member0:=0,member1:=0,member2:=0,member3:=0,member4:=0,member5:=0,member6:=0,member7:=0,member8:=0,member9:=0))]	Numeric data
	Temp	STRING[256]	''	Temporary data
	RS_instance	RS		
	FileOpen_instance	FileOpen		
	FilePuts_instance	FilePuts		
	FileClose_instance	FileClose		

External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag



Determine if SD Memory Card instruction execution is completed.
FileClose_instance.Done
OperatingEnd
FileOpen_instance.Error
FilePuts_instance.Error
FileClose_instance.Error
Accept trigger.
Trigger
_Card1Ready
RS_instance
RS
Set
Q1
Reset1
OperatingEnd
Operating
Initialize row index.
Operating
Inline ST
1 Index:=INT#0;
Execute FileOpen instruction.
Operating
FileOpen_instance
FileOpen
Execute
Done
'ABC.csv'
FileName
Busy
_RDWR_CREATE
Mode
Error
ErrorID
FileID
Fid
FileOpen_instance.Error
Inline ST
1 // Processing after error end
2 ;


Create a text string for one line.
Operating
FileOpen_instance.Done
FilePuts_instance.Busy
FileClose_instance.Busy
AddDelimiter
EN
ENO
myArray[Index]
In
_COMMA
Delimiter
Temp
CONCAT
EN
ENO
Temp
In1
'rl'
In2
StrDat
Write a text string for one line to the file.
Operating
FileOpen_instance.Done
FilePuts_instance.Busy
FileClose_instance.Busy
FilePuts_instance
FilePuts
Execute
Done
Fid
FileID
Busy
StrDat
In
Error
ErrorID
FilePuts_instance.Error
Inline ST
1 // Processing after error end
2 ;
Increment the line index.
Operating
FilePuts_instance.Done
Inc
EN
ENO
InOut
Index
Execute the FileClose instruction after 100 lines are written.
Operating
>
EN
Index
In1
INT#99
In2
FileClose_instance
FileClose
Execute
Done
Fid
FileID
Busy
Error
ErrorID
FileClose_instance.Error
Inline ST
1 // Processing after error end
2 ;
Operating
FileClose_instance.Done
Inline ST
1 // Processing after normal end
2 ;

ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	False	Execution condition
	LastTrigger	BOOL	False	Value of *Trigger* from previous task period
	OperatingStart	BOOL	False	Processing started
	Operating	BOOL	False	Processing
	Stage	INT	0	Stage change
	Index	INT	0	Index
	Fid	DWORD	16#0	File ID
	StrDat	STRING[256]	''	Text string data
	myArray	ARRAY[0..99] OF myStruct	[100((member0:=0,member1:=0,member2:=0,member3:=0,member4:=0,member5:=0,member6:=0,member7:=0,member8:=0,member9:=0))]	Numeric data
	Temp	STRING[256]	''	Temporary data
	FileOpen_instance	FileOpen		
	FilePuts_instance	FilePuts		
	FileClose_instance	FileClose		

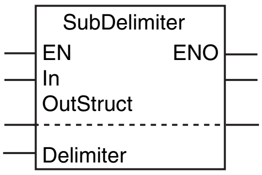
External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag

```
// Start sequence when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Card1Ready=TRUE) ) THEN
    OperatingStart:=TRUE;
    Operating     :=TRUE;
END_IF;
LastTrigger:=Trigger;

// Initialize instance.
IF (OperatingStart=TRUE) THEN
    FileOpen_instance(Execute:=FALSE);
    FilePuts_instance(Execute:=FALSE);
    FileClose_instance(Execute:=FALSE);
    Stage         :=INT#1;
    Index         :=INT#0;       // Initialize row index.
    OperatingStart:=FALSE;
END_IF;

// Execute instruction.
IF (Operating=TRUE) THEN
    CASE Stage OF
    1 :                   // Open file.
        FileOpen_instance(
            Execute :=TRUE,
            FileName:='ABC.csv',       // File name
            Mode    :=_RDWR_CREATE,    // Read file
            FileID  =>Fid);            // File ID

        IF (FileOpen_instance.Done=TRUE) THEN
```

```
            Stage:=INT#2;        // Normal end
        END_IF;

        IF (FileOpen_instance.Error=TRUE) THEN
            Stage:=INT#99;       // Error end
        END_IF;

    2 :                  // Create a text string for one line.
        StrDat:='';

        Temp :=AddDelimiter(myArray[Index],_COMMA);
        StrDat:=CONCAT(In1:=Temp, In2:='$r$l');

        Stage:=INT#3;

    3 :                  // Write text string.
        FilePuts_instance(
            Execute:=TRUE,
            FileID :=Fid,
            In     :=StrDat);

        IF (FilePuts_instance.Done=TRUE) THEN
            Index:=Index+INT#1;

            IF (Index>INT#99) THEN   // If 100 lines were written...
                Stage:=INT#4;
            ELSE
                FilePuts_instance(Execute:=FALSE);
                Stage:=INT#2;
            END_IF;
        END_IF;

        IF (FilePuts_instance.Error=TRUE) THEN
            Stage:=INT#99;       // Error end
        END_IF;

    4 :                  // Close file.
        FileClose_instance(
            Execute:=TRUE,
            FileID :=Fid);            // File ID

        IF (FileClose_instance.Done=TRUE) THEN
            Operating:=FALSE;    // Normal end
        END_IF;

        IF (FileClose_instance.Error=TRUE) THEN
            Stage:=INT#99;       // Error end
        END_IF;

    99 :                 // Processing after error end
        Operating:=FALSE;
    END_CASE;
END_IF;
```

SubDelimiter

The SubDelimiter instruction reads delimited data from a text string and stores the results as the values of the members of a structure.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SubDelimiter	Get Text Strings Minus Delimiters	FUN		Out:=SubDelimiter(In, OutStruct, Delimiter);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Input text string	Input	Delimited text string to convert to the values of the members of a structure	1,986 bytes max. (1,985 single-byte alphanumeric characters plus the final NULL character)	---	''
Delimiter	Delimiter		Delimiter	_COMMA, _TAB, _SEMICOLON, _SPACE	---	_COMMA
OutStruct	Storage structure	In-out	Structure to store results of data conversion	8,192 bytes max.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				OK
Delimiter	Refer to *Function* for the enumerators for the enumerated data type _eDELIMITER.																			
OutStruct	Structure																			
Out	OK																			

Function

The SubDelimiter instruction converts the delimited text string data from input text string *In* and stores the results as the values of the members of storage structure *OutStruct*. The text strings are delimited with delimiter *Delimiter*. The results are stored as the values of the members in order from the start of the structure.

The data type of *Delimiter* is enumerated type _eDELIMITER. The meanings of the enumerators are as follows:

Enumerator	Meaning
_COMMA	‘,’ (comma)
_TAB	‘$T’ (tab)
_SEMICOLON	‘;’(semicolon)
_SPACE	‘ ‘(blank character)

If the number of text strings that are delimited in *In* exceeds the number of members of *OutStruct*, the remaining data is ignored.

If the number of text strings that are delimited in *In* is less than the number of members of *OutStruct*, the values of the remaining members are not changed.

If a member of *OutStruct* is a structure and there is not sufficient data in In for all of the members of that structure, the data is still stored as far as possible.

If a member of *OutStruct* is an array and there is not sufficient data in In for all of the elements of that array, the data is still stored as far as possible.

The delimited data in *In* consists of STRING data. The STRING data is converted according to data types of the members of *OutStruct*, as described next.

● Boolean Data

If the STRING data is ‘FALSE’ or ‘0’, it is converted to FALSE. If the STRING data is ‘TRUE’ or ‘1’, it is converted to TRUE.

The following are exceptions.

- Any continuous ‘0’ characters before ‘0’ or ‘1’ are ignored.
- ‘FALSE’ and ‘TRUE’ are not case sensitive.

Conversion is not possible if the STRING data is not ‘FALSE’, ‘TRUE’, ‘0’, or ‘1’.

● Bit String Data

The conversion rules are the same as for the STRING_TO_** (Text String-to-Bit String Conversion Group) instruction (page 2-301). Conversion is not possible if the data does not express a hexadecimal number.

● Integer Data

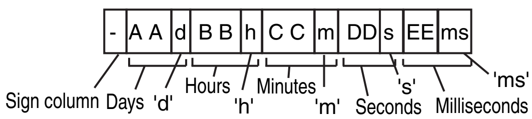
The conversion rules are the same as for the STRING_TO_** (Text String-to-Integer Conversion Group) instruction (page 2-299). Conversion is not possible if the data does not express an integer number.

● Real Number Data

The conversion rules are the same as for the STRING_TO_** (Text String-to-Real Number Conversion Group) instruction (page 2-303). Conversion is not possible if the data does not express a real number.

● Duration Data

Data with the following structure is converted to a duration.



Item	Description
Sign column	If there is a ‘+’ (positive) or if there is no sign column, the value of the member will be positive. If there is a ‘–’ (negative), the value of the member will be negative.
Days	The value of AA is truncated after the 11th digit below the decimal point.
Hours	The value of BB is truncated after the 11th digit below the decimal point.
Minutes	The value of CC is truncated after the 10th digit below the decimal point.
Seconds	The value of DD is truncated after the 9th digit below the decimal point.
Milliseconds	The value of EE is truncated after the 6th digit below the decimal point.

Note 1 Any ‘ ‘ (blank characters) before the sign column, days, hours, minutes, seconds, or milliseconds are ignored.

2 If any characters in the values of AA, BB, CC, DD, or EE are separated with a single ‘_’ (underbar), the underbar is ignored.

3 Even if the value of the days, hours, minutes, seconds, or milliseconds is a real number with a ‘.’ (period), the data can still be converted.

4 If the days, hours, minutes, seconds, or milliseconds is included in the data, conversion is possible even if the other items are omitted.

5 Even if there is a ‘0’ before the value of the days, hours, minutes, seconds, or milliseconds, the data can still be converted.

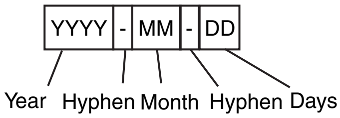
Conversion is not possible in the following cases.

- The data is not in the above structure.
- There is an ‘_’ (underbar) between the sign column and the days.
- ‘.’ (periods) or ‘_’ (underbars) appear consecutively.

For example, if the STRING data is ‘–0.5d48h0.123456789ms’, the value of the member will be T#–2d12h0m0s0.123456ms(T#–216000000.123456ms).

● Date Data

Data with the following structure is converted to a date.



The following are exceptions.

- Any ‘ ‘ (blank characters) before the year, month, or day are ignored.
- If any characters in the values of the year, month, or day are separated with a single ‘_’ (underbar), the underbar is ignored.
- Even if there is a ‘0’ before the value of the year, month, or day, the data can still be converted.

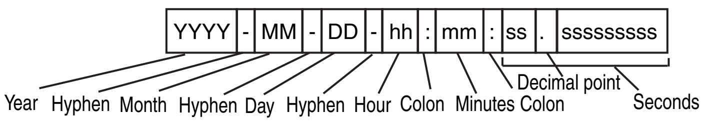
Conversion is not possible in the following cases.

- The data is not in the above structure.
- The date does not exist.

For example, if the STRING data is ‘2000-1-01’, the value of the member will be D#2000-01-01.

● Date and Time Data

Data with the following structure is converted to a duration.



Item	Description
Year, month, and day	This is the year, month, and day that express the date.
Hour	The range of the value is 0 to 23.
Minutes	The range of the value is 0 to 59.
Seconds	The range of the value is 0 to 59.999999999. If the value is an integer, a decimal point is not required.
Hyphens and colons	These are always required.

Note 1 Any ‘ ‘ (blank characters) before the year, month, day, hour, minutes, or seconds are ignored.

2 If any characters in the values of the year, month, day, hour, minutes, or seconds are separated with a single ‘_’ (underbar), the underbar is ignored.

3 Even if there is a ‘0’ before the value of the year, month, day, hour, minutes, or seconds, the data can still be converted.

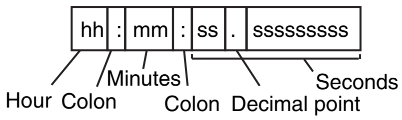
Conversion is not possible in the following cases.

- The data is not in the above structure.
- The date does not exist.

For example, if the STRING data is ‘2000-01-23-4:56:07.89’, the value of the member will be DT#2000-01-23-04:56:07.89.

● Time of Day Data

Data with the following structure is converted to a time of day.



Item	Description
Hour	The range of the value is 0 to 23.
Minutes	The range of the value is 0 to 59.
Seconds	The range of the value is 0 to 59.999999999. If the value is an integer, a decimal point (‘.’ (period)) is not required.
Colons	These are always required.

Note 1 Any ‘ ‘(blank characters) before the hour, minutes, or seconds are ignored.

2 If any characters in the values of hour, minutes, or seconds are separated with a single ‘_’ (underbar), the underbar is ignored.

3 Even if there is a ‘0’ before the value of the hour, minutes, or seconds, the data can still be converted.

Conversion is not possible in the following cases.

- The data is not in the above structure.
- ‘.’ (periods) or ‘_’ (underbars) appear consecutively.

For example, if the STRING data is ‘12:23:34.567’, the value of the member will be TOD#12:23:34.567.

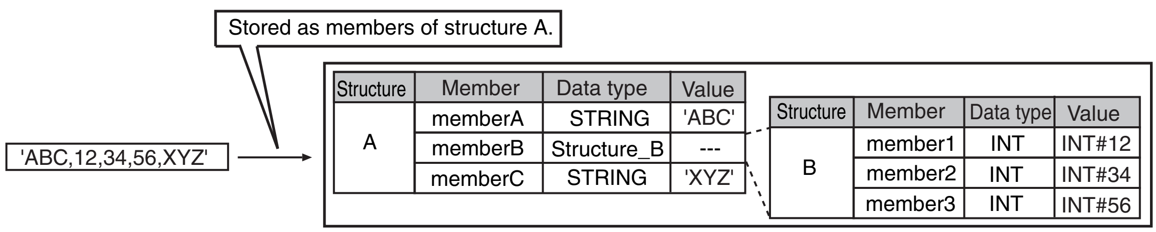
● Text String Data

The value of the member will be the data with a NULL character added to the end. However, conversion is not possible if the text string exceeds the size of the member.

For example, if the STRING data is ‘ABC’ without a NULL character at the end, the value of the member will be ‘ABC’ with a NULL character at the end.

● Structure Data

The STRING data is converted according to the conversion rules for the data types of the members. The data is converted in order from the start and stored as the values of the members of the structure down to the nesting levels that are not structures. For example, if a member of structure A is *Structure_B*, the conversion works as shown below.



● Enumeration Data

STRING data that expresses a DINT variable is converted to an enumerator of the enumeration. The same rules as for integers are used to convert to DINT data, the value of the DINT data is taken as the value of the enumeration, and that value is converted to the corresponding enumerator. However, conversion is not possible if the STRING data does not express a DINT value. For example, assume that an enumeration *Color* has three enumerators: *red*, *yellow*, and *green*. The numbers associated with these enumerators are as follows: *red* = 1, *yellow* = 2, *green* = 3. If the data is '3', the value of the member will be *green*.

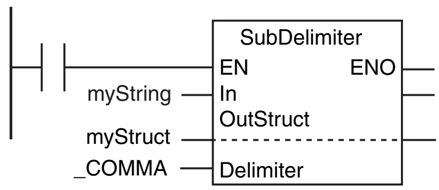
● Array Data

Each delimited data is converted to the value of an element. The conversion rules for the data type of the array are used. Conversion is possible only if the members are one-dimensional arrays. For example, assume that a member is the *myString[0..3]* BYTE array. If the comma-delimited text string 'AA,BB,CC,DD' is converted to the elements of the array, *myString[0]* will be BYTE#16#AA, *myString[1]* will be BYTE#16#BB, *myString[2]* will be BYTE#16#CC, and *myString[3]* will be BYTE#16#DD.

Notation Example

The following notation reads comma-delimited data from the myString text string and stores the text strings as the values of the members of the myStruct structure.

LD



ST

```
SubDelimiter(myString,myStruct,_COMMA);
```

myString 'XYZ,1234,aB,3.14'

Structure	Member	Data type	Value
myStruct	member1	STRING	' '
	member2	INT	INT#0
	member3	WORD	WORD#16#0000
	member4	REAL	REAL#0

Data from myString is stored as the values of the members of myStruct.

Structure	Member	Data type	Value
myStruct	member1	STRING	'XYZ'
	member2	INT	INT#1234
	member3	WORD	WORD#16#00AB
	member4	REAL	REAL#3.14e+0

Additional Information

- You can combine this instruction with the *FileGets instruction* (page 2-1293) to easily read values from specified CSV files in an SD Memory Card. Refer to *Sample Programming* for an application example.
- Use this instruction to return a text string that was converted with the *AddDelimiter instruction* (page 2-577) to structure data.

Precautions for Correct Use

- If there is more than one consecutive delimiter in *In*, the delimited data will not exist. If the delimited data does not exist, the value of the member of *OutStruct* will be undefined.
- Do not use the delimiter in *In* for anything other than the delimiter. If you use the delimiter for any other purpose, the instruction will still treat it as a delimiter.

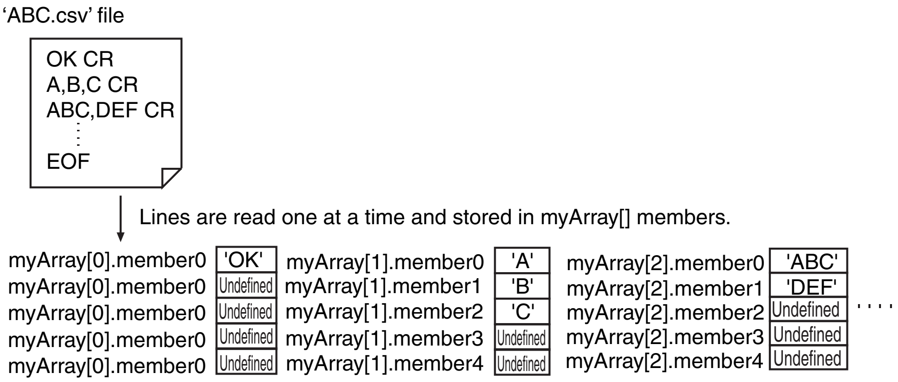
- If there is a STRING member in *OutStruct*, do not attach a final NULL character to the corresponding data in *In*. If you use a NULL character anywhere except at the end of *In*, only the text string through the first NULL character will be converted.
- If there is an enumeration in *OutStruct*, make sure that the corresponding data in *In* is a value that is defined as an enumerator. An error will not occur even if the value of the enumerated variable is not a value that is defined as an enumerator.
- An error occurs in the following cases. *ENO* changes to FALSE and the values in *OutStruct* will be undefined.
  - Conversion to a value with the data type of the member of *OutStruct* is not possible.
  - The conversion result exceeds the valid range of the value of the data type of the member of *Out-Struct*.
  - A member of *OutStruct* is an array with more than one dimension.
  - A member of *OutStruct* is a union.
  - The size of *OutStruct* exceeds 8,192 bytes.

Version Information

A CPU Unit with unit version 1.02 or later and Sysmac Studio version 1.03 or higher are required to use this instruction.

Sample Programming

Here, multiple lines of text strings that are separated by carriage returns (i.e., CR codes) are stored in a file named ‘ABC.csv.’ The text string on each line is delimited by commas. Text strings are read from this file one line at a time, and the comma-delimited data is stored as the values of the members of the *myArray[]* array variables in the *mystruct* structure from the start of the structure. The *myStruct* structure has five members that are STRING variables.
Processing ends when the data is read to the end of the file (i.e., when it is read to the EOF code).



The processing procedure is as follows:

1. The FileOpen instruction is used to open the file ‘ABC.csv.’
2. The FileGets instruction is used to read one line from the file.
3. The SubDelimiter is used to store comma-delimited text strings as the values of the *myArray[]* members.
4. Steps 2 and 3 are repeated until the EOF (end of file).
5. The FileClose instruction is used to close the file.

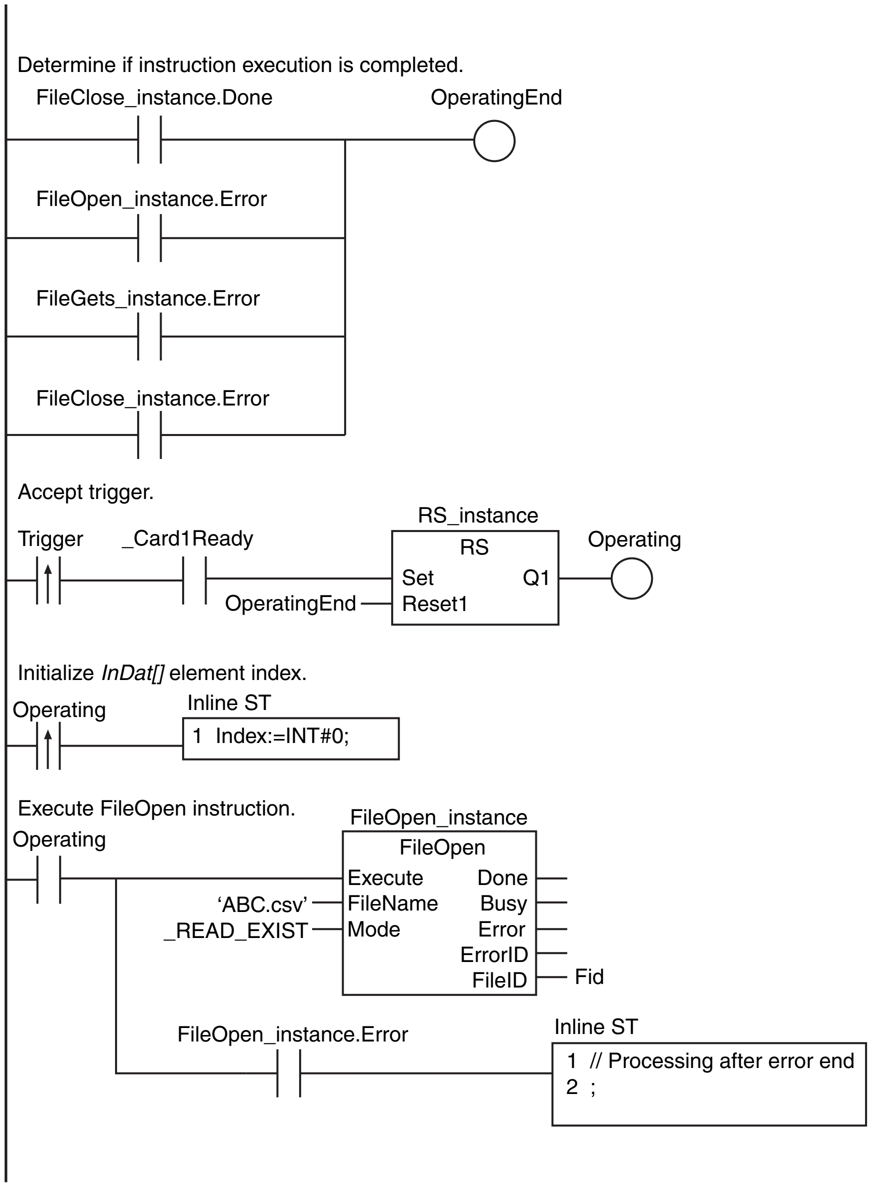
Data Type Definition

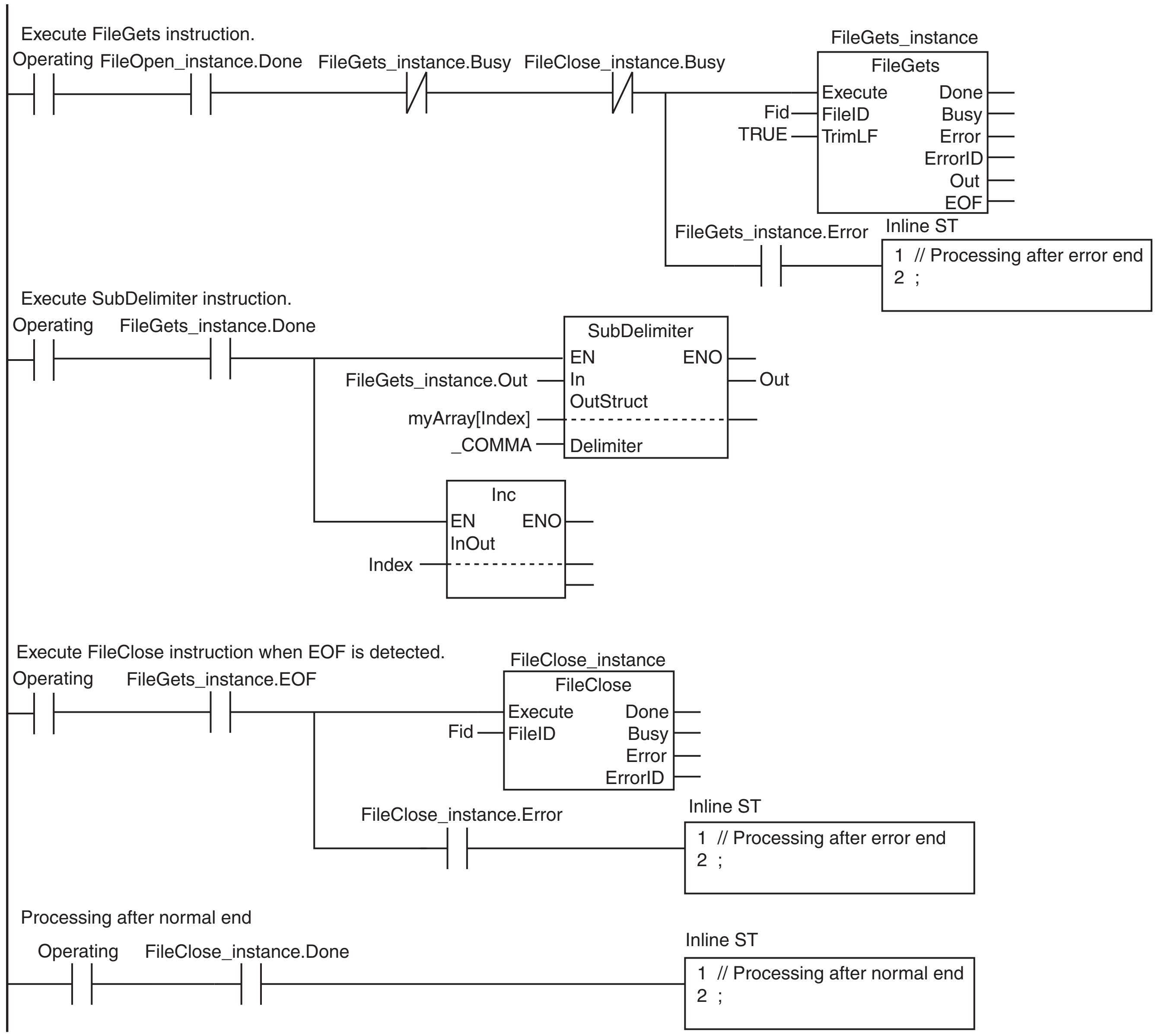
Name	Data type	Comment
myStruct	STRUCT	Structure
member0	STRING	Member
member1	STRING	Member
member2	STRING	Member
member3	STRING	Member
member4	STRING	Member

LD

Internal Variables	Variable	Data type	Initial value	Comment
	OperatingEnd	BOOL	False	Processing completed
	Trigger	BOOL	False	Execution condition
	Operating	BOOL	False	Processing
	Index	INT	0	myArray[] element index
	Fid	DWORD	16#0	File ID
	myArray	ARRAY[0..999] OF myStruct	[1000((member0:='',member1:='',member2:='',member3:='',member4:=''))]	Integer data
	RS_instance	RS		
	FileOpen_instance	FileOpen		
	FileGets_instance	FileGets		
	FileClose_instance	FileClose		

External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag


Determine if instruction execution is completed.
FileClose_instance.Done
OperatingEnd
FileOpen_instance.Error
FileGets_instance.Error
FileClose_instance.Error
Accept trigger.
Trigger
_Card1Ready
RS_instance
RS
Set
Q1
OperatingEnd
Reset1
Operating
Initialize InDat[] element index.
Operating
Inline ST
1 Index:=INT#0;
Execute FileOpen instruction.
FileOpen_instance
Operating
FileOpen
Execute
Done
'ABC.csv'
FileName
Busy
_READ_EXIST
Mode
Error
ErrorID
FileID
Fid
FileOpen_instance.Error
Inline ST
1 // Processing after error end
2 ;



Execute FileGets instruction.
Operating
FileOpen_instance.Done
FileGets_instance.Busy
FileClose_instance.Busy
FileGets_instance
FileGets
Execute
Done
Fid
FileID
Busy
TRUE
TrimLF
Error
ErrorID
Out
EOF
FileGets_instance.Error
Inline ST
1 // Processing after error end
2 ;
Execute SubDelimiter instruction.
Operating
FileGets_instance.Done
SubDelimiter
EN
ENO
FileGets_instance.Out
In
Out
OutStruct
myArray[Index]
_COMMA
Delimiter
Inc
EN
ENO
InOut
Index
Execute FileClose instruction when EOF is detected.
Operating
FileGets_instance.EOF
FileClose_instance
FileClose
Execute
Done
Fid
FileID
Busy
Error
ErrorID
FileClose_instance.Error
Inline ST
1 // Processing after error end
2 ;
Processing after normal end
Operating
FileClose_instance.Done
Inline ST
1 // Processing after normal end
2 ;

ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	False	Execution condition
	LastTrigger	BOOL	False	Value of *Trigger* from previous task period
	OperatingStart	BOOL	False	Processing started
	Operating	BOOL	False	Processing
	myArray	ARRAY[0..999] OF myStruct	[1000((member0:='',member1:='',member2:='',member3:='',member4:=''))]	Integer data
	Stage	INT	0	Stage change
	Index	INT	0	myArray[] element index
	Fid	DWORD	16#0	File ID
	FileOpen_instance	FileOpen		
	FileGets_instance	FileGets		
	FileClose_instance	FileClose		

External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag

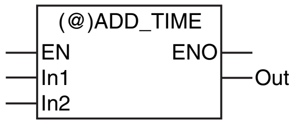
```
// Start sequence when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Card1Ready=TRUE) ) THEN
    OperatingStart:=TRUE;
    Operating     :=TRUE;
END_IF;
LastTrigger:=Trigger;

// Initialize instance.
IF (OperatingStart=TRUE) THEN
    FileOpen_instance(Execute:=FALSE);
    FileGets_instance(Execute:=FALSE);
    FileClose_instance(Execute:=FALSE);
    Stage         :=INT#1;
    Index         :=INT#0;
    OperatingStart:=FALSE;
END_IF;

// Execute instruction.
IF (Operating=TRUE) THEN
    CASE Stage OF
    1 :                  // Open file.
        FileOpen_instance(
            Execute :=TRUE,
            FileName:='ABC.csv',     // File name
            Mode    :=_READ_EXIST,   // Read file
            FileID  =>Fid);          // File ID

        IF (FileOpen_instance.Done=TRUE) THEN
            Stage:=INT#2;       // Normal end
        END_IF;

        IF (FileOpen_instance.Error=TRUE) THEN
```

```
            Stage:=INT#99;      // Error end
        END_IF;



    2 :                   // Read text string.
        FileGets_instance(
            Execute:=TRUE,
            FileID :=Fid,
            TrimLF :=TRUE);

        IF (FileGets_instance.Done=TRUE) THEN
            // Store the text strings that were read as the values of the
myArray[] member.
            SubDelimiter(FileGets_instance.Out,myArray[Index],_COMMA);
            Index:=Index+INT#1;

            // Reached end of file.
            IF (FileGets_instance.EOF=TRUE) THEN
                Stage:=INT#3;   // Normal end
            ELSE
                FileGets_instance(Execute:=FALSE);
            END_IF;
        END_IF;

        IF (FileGets_instance.Error=TRUE) THEN
            Stage:=INT#99;      // Error end
        END_IF;

     3 :                  // Close file.
        FileClose_instance(
            Execute:=TRUE,
            FileID :=Fid);           // File ID

        IF (FileClose_instance.Done=TRUE) THEN
            Operating:=FALSE;   // Normal end
        END_IF;

        IF (FileClose_instance.Error=TRUE) THEN
            Stage:=INT#99;      // Error end
        END_IF;

      99 :                // Processing after error end
         Operating:=FALSE;
    END_CASE;
END_IF;
```

Time and Time of Day Instructions

ADD_TIME

The ADD_TIME instruction adds two times.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ADD_TIME	Add Time	FUN		Out:=ADD_TIME(In1, In2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Add time 1	Input	Add time 1	Depends on data type.	ns	T#0s
In2	Add time 2		Add time 2			
Out	Total time	Output	Total time	Depends on data type.	ns	---

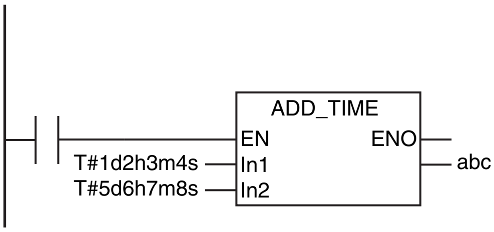
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																OK				
In2																OK				
Out																OK				

Function

The ADD_TIME instruction adds two times, *In1* and *In2*. The result of addition in *Out* is also a time.

The following example is for when *In1* is T#1d2h3m4s and *In2* is T#5d6h7m8s.

LD



ST

```
abc:=ADD_TIME(T#1d2h3m4s, T#5d6h7m8s);
```

	In1	T#1d2h3m4s
+	In2	T#5d6h7m8s
	Out=abc	T#6d8h10m12s

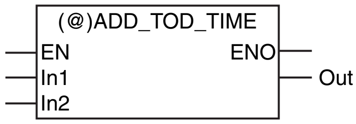
Precautions for Correct Use

An error will not occur even if the addition result exceeds the valid range of *Out*.

- T#106751d_23h_47m_16s_854.775807ms + T#0.000001ms
  → T#-106751d_23h_47m_16s_854.775808ms
- T#-106751d_23h_47m_16s_854.775808ms + T#-0.000001ms
  → T#106751d_23h_47m_16s_854.775807ms

ADD_TOD_TIME

The ADD_TOD_TIME instruction adds a time to a time of day.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ADD_TOD_TIME	Add Time to Time of Day	FUN		Out:=ADD_TOD_TIME(In1, In2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Add time of day	Input	Add time of day	Depends on data type.	Hour, minutes, seconds	TOD#0:0:0
In2	Add time		Add time		ns	T#0s
Out	Resulting time of day	Output	Resulting time of day	Depends on data type.	Hour, minutes, seconds	---

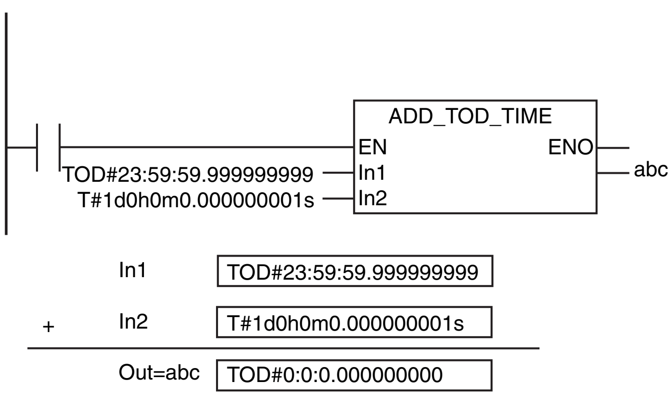
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																		OK		
In2																OK				
Out																		OK		

Function

The ADD_TOD_TIME instruction adds a time, *In2,* to a time of day *In1.* The result of addition in *Out* is also a time of day.

The following example is for when *In1* is TOD#23:59:59.999999999 and *In2* is T#1d0h0m0.000000001s.

LD



ST

```
abc:=ADD_TOD_TIME(TOD#23:59:59.999999999,
                  T#1d0h0m0.000000001s);
```

Precautions for Correct Use

An error will not occur even if the addition result exceeds the valid range of *Out*.

- TOD#23:59:59.999999999 + T#0.000001ms → TOD#0:0:0.000000000
- TOD#0:0:0.000000000 + T#-0.000001ms → TOD#23:59:59.999999999

ADD_DT_TIME

The ADD_DT_TIME instruction adds a time to a date and time.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ADD_DT_TIME	Add Time to Date and Time	FUN	(@)ADD_DT_TIME EN ENO In1 Out In2	Out:=ADD_DT_TIME(In1, In2);

Variables

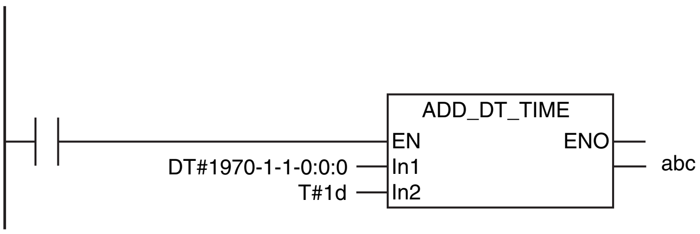
Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Add date and time	Input	Add date and time	Depends on data type.	Year, month, day, hour, minutes, seconds	DT#1970-1-1-0:0:0
In2	Add time		Add time		ns	T#0s
Out	Addition result date and time	Output	Addition result date and time	Depends on data type.	Year, month, day, hour, minutes, seconds	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																			OK	
In2																OK				
Out																			OK	

Function

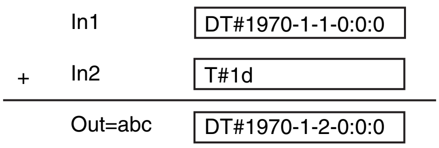
The ADD_DT_TIME instruction adds a time, *In2,* to a date and time *In1.* The result of addition in *Out* is also a date and time. Leap years are also accounted for. The following example is for when *In1* is DT#1970-1-1-0:0:0 and *In2* is T#1d.

LD



ST

abc:=ADD_DT_TIME(DT#1970-1-1-0:0:0, T#1d);



Related System-defined Variables

Name	Meaning	Data type	Description
_CurrentTime	System Time of Day	DT	The time of day from the system clock. The number of seconds from 00:00:00 on January 1,1970.

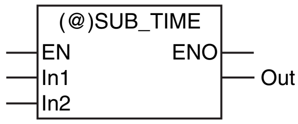
Precautions for Correct Use

An error will not occur even if the addition result exceeds the valid range of *Out*.

- DT#2554-7-21-23:34:33.709551615 + T#0.000001ms → DT#1970-1-1-0:0:0
- DT#1970-1-1-0:0:0 + T#-0.000001ms → DT#2554-7-21-23:34:33.709551615

SUB_TIME

The SUB_TIME instruction subtracts one time from another.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SUB_TIME	Subtract Time	FUN		Out:=SUB_TIME(In1, In2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Original time	Input	Original time	Depends on data type.	ns	T#0s
In2	Time to subtract		Time to subtract			
Out	Resulting time	Output	Resulting time	Depends on data type.	ns	---

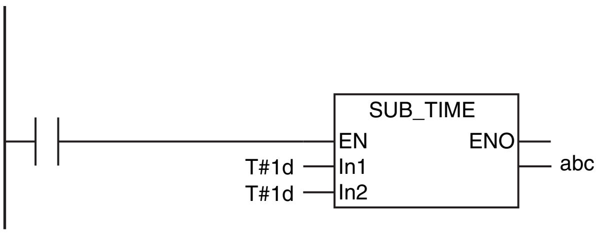
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																OK				
In2																OK				
Out																OK				

Function

The SUB_TIME instruction subtracts a time *In2* from another time *In1.* The result of subtraction in *Out* is also a time.

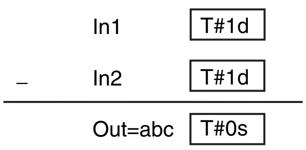
The following example is for when *In1* and *In2* are T#1d.

LD



ST

abc:=SUB_TIME(T#1d, T#1d);



Precautions for Correct Use

An error will not occur even if the subtraction result exceeds the valid range of *Out*.

- T#106751d_23h_47m_16s_854.775807ms – T#-0.000001ms
  → T#-106751d_23h_47m_16s_854.775808ms
- T#-106751d_23h_47m_16s_854.775808ms – T#0.000001ms
  → T#106751d_23h_47m_16s_854.775807ms

SUB_TOD_TIME

The SUB_TOD_TIME instruction subtracts a time from a time of day.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SUB_TOD_TIME	Subtract Time from Time of Day	FUN	(@)SUB_TOD_TIME EN ENO In1 Out In2	Out:=SUB_TOD_TIME(In1, In2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Time of day	Input	Time of day	Depends on data type.	Hour, minutes, seconds	TOD#0:0:0
In2	Time to subtract		Time to subtract		ns	T#0s
Out	Resulting time of day	Output	Resulting time of day	Depends on data type.	Hour, minutes, seconds	---

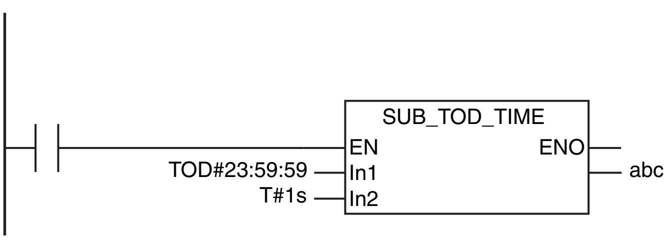
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																		OK		
In2																OK				
Out																		OK		

Function

The SUB_TOD_TIME instruction subtracts a time *In2* from a time of day *In1*. The result of subtraction in *Out* is also a time of day.

The following example is for when *In1* is TOD#23:59:59 and *In2* is T#1s.

LD



ST

abc:=SUB_TOD_TIME(TOD#23:59:59, T#1s);

	In1	TOD#23:59:59
−	In2	T#1s
	Out=abc	TOD#23:59:58

Precautions for Correct Use

An error will not occur even if the subtraction result exceeds the valid range of *Out*.

- TOD#23:59:59.999999999 – T#-0.000001ms → TOD#0:0:0
- TOD#0:0:0 – T#0.000001ms → TOD#23:59:59.999999999

SUB_TOD_TOD

The SUB_TOD_TOD instruction subtracts a time of day from another time of day.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SUB_TOD_TOD	Subtract Time of Day	FUN	(@)SUB_TOD_TOD EN ENO In1 Out In2	Out:=SUB_TOD_TOD(In1, In2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Time of day 1	Input	Time of day 1	Depends on data type.	Hour, minutes, seconds	TOD#0:0:0
In2	Time of day 2		Time of day 2			
Out	Resulting time	Output	Resulting time	Depends on data type.	ns	---

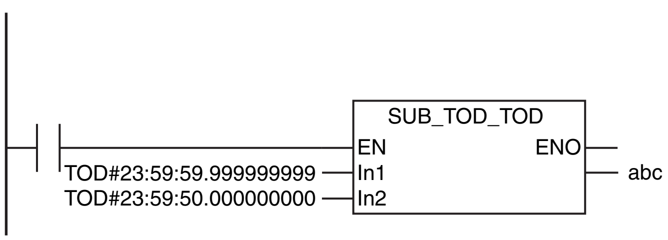
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																		OK		
In2																		OK		
Out																OK				

Function

The SUB_TOD_TOD instruction subtracts time of day *In2* from time of day *In1*. The result of subtraction in *Out* is a time.

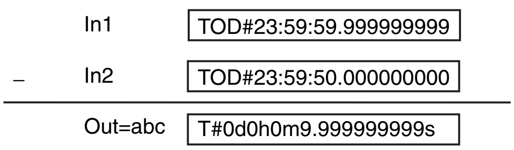
The following example is for when *In1* is TOD#23:59:59.999999999 and *In2* is TOD#23:59:50.000000000.

LD



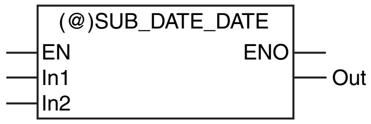
ST

```
abc:=SUB_TOD_TOD(TOD#23:59:59.999999999,
                 TOD#23:59:50.000000000);
```



SUB_DATE_DATE

The SUB_DATE_DATE instruction subtracts another date from another date.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SUB_DATE_DATE	Subtract Date	FUN		Out:=SUB_DATE_DATE(In1, In2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Date 1	Input	Date 1	Depends on data type.	Year, month, day	D#1970-1-1
In2	Date 2		Date 2			
Out	Resulting time	Output	Resulting time	Depends on data type.	ns	---

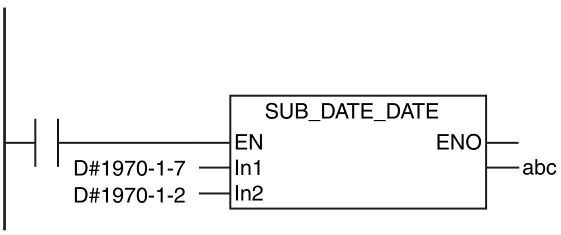
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																	OK			
In2																	OK			
Out																OK				

Function

The SUB_DATE_DATE instruction subtracts date *In2* from date *In1*. The result of subtraction in *Out* is a time.

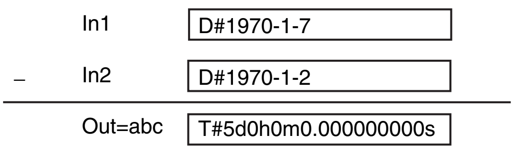
The following example is for when *In1* is D#1970-1-7 and *In2* is D#1970-1-2.

LD



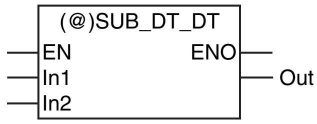
ST

```
abc:=SUB_DATE_DATE(D#1970-1-7, D#1970-1-2);
```



SUB_DT_DT

The SUB_DT_DT instruction subtracts another date and time from another date and time.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SUB_DT_DT	Subtract Date and Time	FUN		Out:=SUB_DT_DT(In1, In2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Date and time 1	Input	Date and time 1	Depends on data type.	Year, month, day, hour, minutes, seconds	DT#1970-1-1-0:0:0
In2	Date and time 2		Date and time 2			
Out	Resulting time	Output	Resulting time	Depends on data type.	ns	---

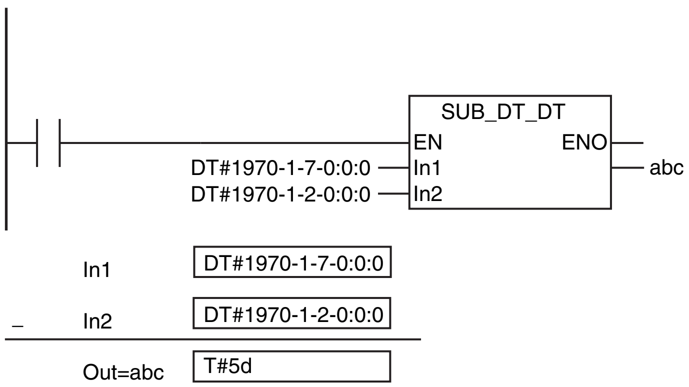
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																			OK	
In2																			OK	
Out																OK				

Function

The SUB_DT_DT instruction subtracts date and time *In2* from date and time *In1.* The result of subtraction in *Out* is a time.

The following example is for when *In1* is DT#1970-1-7-0:0:0 and *In2* is DT#1970-1-2-0:0:0.

LD



ST

```
abc:=SUB_DT_DT(DT#1970-1-7-0:0:0,
               DT#1970-1-2-0:0:0);
```

Related System-defined Variables

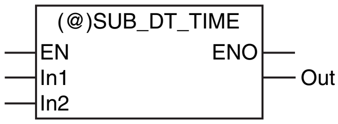
Name	Meaning	Data type	Description
_CurrentTime	System Time of Day	DT	The time of day from the system clock. The number of seconds from 00:00:00 on January 1,1970.

Precautions for Correct Use

If the processing result exceeds the valid range of *Out*, *Out* will contain an illegal value.

SUB_DT_TIME

The SUB_DT_TIME instruction subtracts a time from a date and time.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SUB_DT_TIME	Subtract Time from Date and Time	FUN		Out:=SUB_DT_TIME(In1, In2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Date and time	Input	Date and time	Depends on data type.	Year, month, day, hour, minutes, seconds	DT#1970-1-1-0:0:0
In2	Time to subtract		Time to subtract		ns	T#0s
Out	Resulting date and time	Output	Resulting date and time	Depends on data type.	Year, month, day, hour, minutes, seconds	---

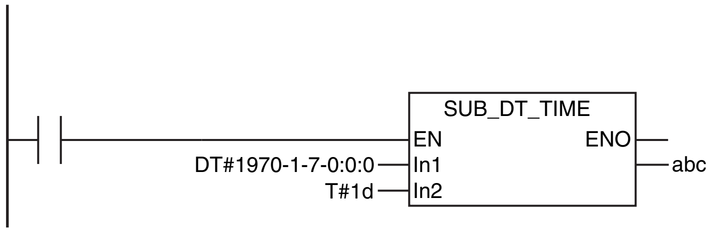
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																			OK	
In2																OK				
Out																			OK	

Function

The SUB_DT_TIME instruction subtracts a time *In2* from a date and time *In1.* The result of subtraction in *Out* is a date and time. Leap years are also accounted for.

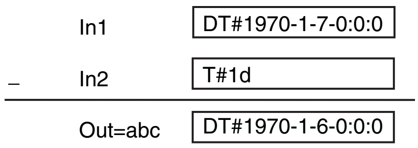
The following example is for when *In1* is DT#1970-1-1-0:0:0 and *In2* is T#1d.

LD



ST

```
abc:=SUB_DT_TIME(DT#1970-1-7-0:0:0, T#1d);
```



Related System-defined Variables

Name	Meaning	Data type	Description
_CurrentTime	System Time of Day	DT	The time of day from the system clock. The number of seconds from 00:00:00 on January 1,1970.

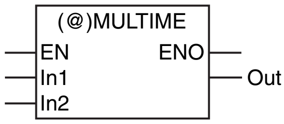
Precautions for Correct Use

An error will not occur even if the subtraction result exceeds the valid range of *Out*.

- DT#2554-7-21-23:34:33.709551615 – T#-0.000001ms → DT#1970-1-1-0:0:0
- DT#1970-1-1-0:0:0 – T#0.000001ms → DT#2554-7-21-23:34:33.709551615

MULTIME

The MULTIME instruction multiplies a time by a specified number.

Instruction	Name	FB/FUN	Graphic expression	ST expression
MULTIME	Multiply Time	FUN		Out:=MULTIME(In1, In2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Original time	Input	Original time	Depends on data type.	ns	T#0s
In2	Multiplier		Multiplier		---	*
Out	Resulting time	Output	Resulting time	Depends on data type.	ns	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

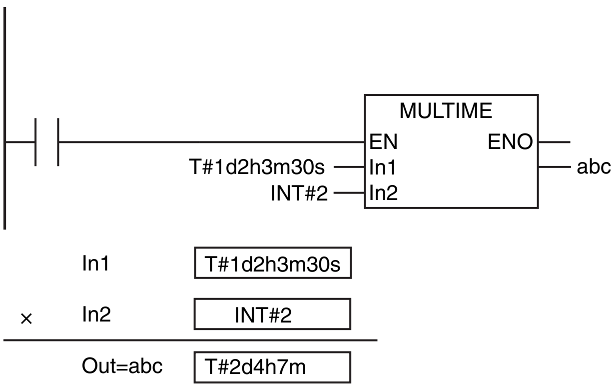
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																OK				
In2						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Out																OK				

Function

The MULTIME instruction multiplies a time *In1* by multiplier *In2*. The result of multiplication in *Out* is also a time.

The following example is for when *In1* is T#1d2h3m30s and *In2* is INT#2.

LD



ST

abc:=MULTIME(T#1d2h3m30s, INT#2);

Precautions for Correct Use

- If *In2* is a real number, the multiplication result is rounded to the nearest nanosecond. The following table shows how values are rounded.

Value below nanosec-onds	Treatment	Examples
Less than 0.5	The value is truncated.	1.49 → 1
0.5	If the ones digit is an even number, the value is trun-cated. If it is an odd number, the value is rounded up.	1.50 → 2 2.50 → 2
Greater than 0.5	The value is rounded up.	1.51 → 2

- If the value of *In2* is 0, positive infinity, negative infinity, or nonnumeric data, the value of *Out* is as shown below.

Value of *In2*	Value of *Out*	
	Other than the right	NX1P2
0.0	T#0s	T#0s
+∞	T#-106751d23h47m16.854775808s	T#-0d0h0m0s1e-6ms
–∞	T#-106751d23h47m16.854775808s	T#-0d0h0m0s1e-6ms
Nonnumeric data	T#-106751d23h47m16.854775808s	T#0s

- An error will not occur even if the multiplication result exceeds the valid range of *Out*.
  - T#53375d_23h_53m_38s_427.387904ms * USINT#2
    → T#-106751d_23h_47m_16s_854.775808ms
  - T#-53375d_23h_53m_38s_427.387905ms * USINT#2
    → T#106751d_23h_47m_16s_854.775806ms

DIVTIME

The DIVTIME instruction divides a time by a specified number.

Instruction	Name	FB/FUN	Graphic expression	ST expression
DIVTIME	Divide Time	FUN	(@)DIVTIME EN ENO In1 Out In2	Out:=DIVTIME(In1, In2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Original time	Input	Original time	Depends on data type.	ns	T#0s
In2	Number to divide by	Input	Number to divide by	Depends on data type.	---	*
Out	Resulting time	Output	Resulting time	Depends on data type.	ns	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

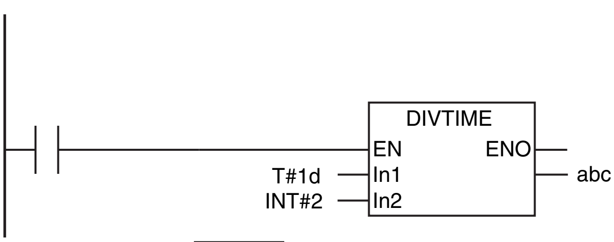
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																OK				
In2						OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
Out																OK				

Function

The DIVTIME instruction divides a time *In1* by a number *In2*. The result of division in *Out* is also a time.

The following example is for when *In1* is T#1d and *In2* is INT#2.

LD



ST

abc:=DIVTIME(T#1d, INT#2);

	In1	T#1d
/	In2	INT#2
	Out=abc	T#12h

Precautions for Correct Use

- If the value of *In2* is 0, positive infinity, negative infinity, or nonnumeric data, the value of *Out* is as shown below.

Value of *In2*	Value of *Out*	
	Other than the right	NX1P2
0.0	T#-106751d23h47m16.854775808s	T#0d_0h_0m_0s_1e-006
+∞	T#0s	T#0s
−∞	T#0s	T#0s
Nonnumeric data	T#-106751d23h47m16.854775808s	T#0s

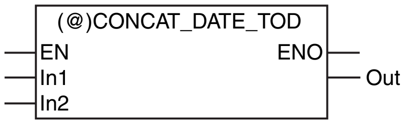
- If *In2* is a real number, there may be error of up to several nanoseconds.
- If *In2* is a real number, the division result is rounded to the nearest nanosecond. The following table shows how values are rounded.

Value below nanoseconds	Description	Example
Less than 0.5	The fractional part is truncated.	1.49 → 1
0.5	If the ones digit is an even number, the value is truncated. If it is an odd number, the value is rounded up.	1.50 → 2 2.50 → 2
Greater than 0.5	The fractional part is rounded up.	1.51 → 2

- An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.
  - *In2* is an integer with a value of 0.

CONCAT_DATE_TOD

The CONCAT_DATE_TOD instruction combines a date and a time of day.

Instruction	Name	FB/FUN	Graphic expression	ST expression
CONCAT_DATE_TOD	Concatenate Date and Time of Day	FUN		Out:=CONCAT_DATE_TOD(In1, In2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In1	Date	Input	Date	Depends on data type.	Year, month, day	D#1970-1-1
In2	Time of day		Time of day		Hour, minutes, seconds	TOD#0:0:0
Out	Combined date and time	Output	Combined date and time	Depends on data type.	Year, month, day, hour, minutes, seconds	---

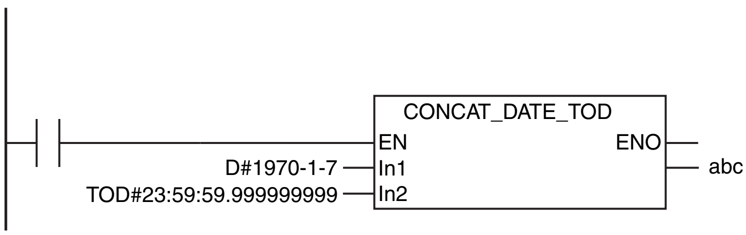
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																	OK			
In2																		OK		
Out																			OK	

Function

The CONCAT_DATE_TOD instruction combines a date *In1* and a time of day *In2.* The result of combining in *Out* is also a date and time.

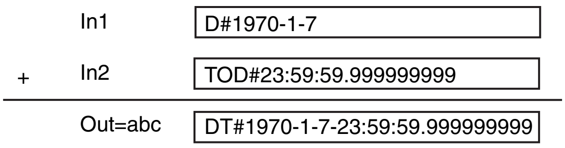
The following example is for when *In1* is D#1970-1-7 and *In2* is TOD#23:59:59.999999999.

LD



ST

```
abc:=CONCAT_DATE_TOD(D#1970-1-7,
                     TOD#23:59:59.999999999);
```



Related System-defined Variables

Name	Meaning	Data type	Description
_CurrentTime	System Time of Day	DT	The time of day from the system clock. The number of seconds from 00:00:00 on January 1,1970.

Precautions for Correct Use

An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.

- The results of combining exceeds the valid range of *Out* (e.g., the value of *In1* is D#2554-7-21 and the value of *In2* is larger than TOD#23:34:33.709551615).

DT_TO_TOD

The DT_TO_TOD instruction extracts the time of day from a date and time.

Instruction	Name	FB/FUN	Graphic expression	ST expression
DT_TO_TOD	Extract Time of Day from Date and Time	FUN	(@)DT_TO_TOD EN ENO In Out	Out:=DT_TO_TOD(In);

Variables

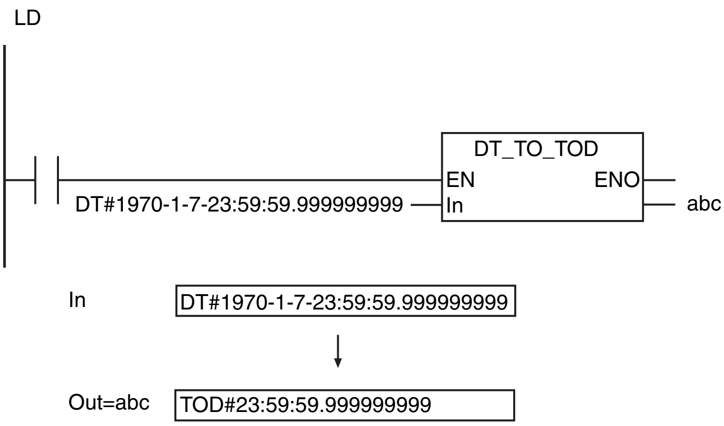
Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Date and time	Input	Date and time	Depends on data type.	Year, month, day, hour, minutes, seconds	DT#1970-1-1-0:0:0
Out	Time of day	Output	Time of day	Depends on data type.	Hour, minutes, seconds	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																			OK	
Out																		OK		

Function

The DT_TO_TOD instruction extracts the time of day from date and time *In*.

The following example is for when *In* is DT#1970-1-7-23:59:59.999999999.



ST

abc:=DT_TO_TOD(DT#1970-1-7-23:59:59.999999999);

Related System-defined Variables

Name	Meaning	Data type	Description
_CurrentTime	System Time of Day	DT	The time of day from the system clock. The number of seconds from 00:00:00 on January 1,1970.

DT_TO_DATE

The DT_TO_DATE instruction extracts the date from a date and time.

Instruction	Name	FB/FUN	Graphic expression	ST expression
DT_TO_DATE	Extract Date from Date and Time	FUN	(@)DT_TO_DATE EN ENO In Out	Out:=DT_TO_DATE(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Date and time	Input	Date and time	Depends on data type.	Year, month, day, hour, minutes, seconds	DT#1970-1-1-0:0:0
Out	Date	Output	Date	Depends on data type.	Year, month, day	---

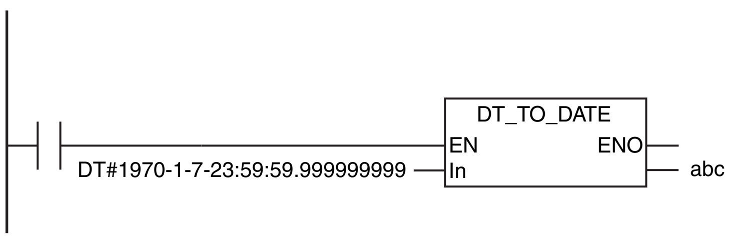
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																			OK	
Out																	OK			

Function

The DT_TO_DATE instruction extracts the date from date and time *In.*

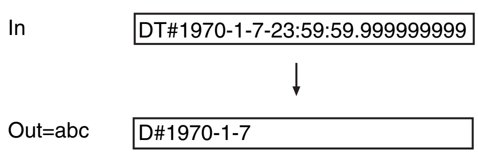
The following example is for when *In* is DT#1970-1-7-23:59:59.999999999.

LD



ST

abc:=DT_TO_DATE(DT#1970-1-7-23:59:59.999999999);

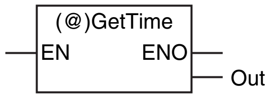


Related System-defined Variables

Name	Meaning	Data type	Description
_CurrentTime	System Time of Day	DT	The time of day from the system clock. The number of seconds from 00:00:00 on January 1,1970.

GetTime

The GetTime instruction reads the current time.

Instruction	Name	FB/FUN	Graphic expression	ST expression
GetTime	Get Time of Day	FUN		Out:=GetTime();

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Out	Current time	Output	Current time	*	Year, month, day, hour, minutes, seconds	---

* The valid range is for any of the following GMTs (Greenwich Mean Times).
The valid range for an NX-series CPU Unit is DT#1970-01-01-00:00:00.000000000 to DT#2069-12-31-23:59:59.999999999 (0:00:000000000 on January 1, 1970 to 23:59:59.999999999 on December 31, 2069).
The valid range for an NJ-series CPU Unit is DT#1970-01-01-00:00:00.000000000 to DT#2106-02-06-23:59:59.999999999 (0:00:000000000 on January 1, 1970 to 23:59:59.999999999 on February 6, 2106).
The valid range for an NY-series Controller is DT#2000-01-01-00:00:00.000000000 to DT#2099-12-31-23:59:59.999999999 (0:00:000000000 on January 1, 2000 to 23:59:59.999999999 on December 31, 2099).

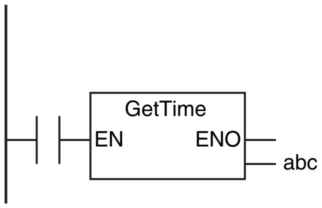
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out																			OK	

Function

The GetTime instruction reads the current time. The current time of day is the time for the set time zone (not Greenwich mean time (GMT)).

The following figure shows a programming example. The current time is assigned to variable *abc.*

LD



ST

```
abc:=GetTime();
```

The GetTime instruction assigns the current time to *abc*.

For 23:59 and 59.999999999 seconds on January 7, 1970

System Time of Day

23:59 and 59.999999999 seconds on January 7, 1970

System time of day is assigned. → Out=abc DT#1970-1-7-23:59:59.999999999

Related System-defined Variables

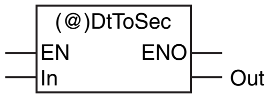
Name	Meaning	Data type	Description
_CurrentTime	System Time of Day	DT	The time of day from the system clock. The number of seconds from 00:00:00 on January 1,1970.

Additional Information

- Use the DtToSec instruction (page 2-628) to convert the current time of day to the system time of day (number of seconds from 00:00:00 on January 1,1970).
- Use the DtToDateStruct instruction (page 2-652) to convert the current time of day to a date (year, month, day, minutes, and seconds).
- Use the GetDayOfWeek instruction (page 2-648) to read the day of the week.

DtToSec

The DtToSec instruction converts a date and time to the number of seconds from 00:00:00 on January 1, 1970.

Instruction	Name	FB/FUN	Graphic expression	ST expression
DtToSec	Convert Date and Time to Seconds	FUN		Out:=DtToSec(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Date and time	Input	Date and time	Depends on data type.	Year, month, day, hour, minutes, seconds	DT#1970-1-1-0:0:0
Out	Seconds	Output	Number of seconds from 00:00:00 on January 1, 1970	0 to 18446744073	Seconds	---

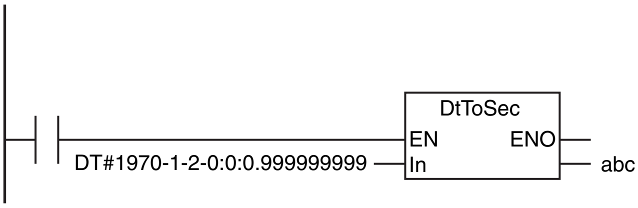
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																			OK	
Out													OK							

Function

The DtToSec instruction converts the date and time in *In* to the number of seconds from 00:00:00 on January 1, 1970. The converted value is in seconds. The value is truncated below the seconds.

The following example is for when *In* is DT#1970-1-2-0:0:0.999999999.

LD



ST

abc:=DtToSec(DT#1970-1-2-0:0:0.999999999);

In DT#1970-1-2-0:0:0.999999999

– DT#1970-1-1-0:0:0.000000000

Out=abc LINT#86400 s

Related System-defined Variables

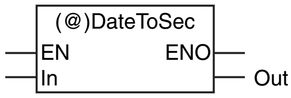
Name	Meaning	Data type	Description
_CurrentTime	System Time of Day	DT	The time of day from the system clock. The number of seconds from 00:00:00 on January 1,1970.

Additional Information

Use the SecToDt instruction (page 2-632) to convert the number of seconds from 00:00:00 on January 1,1970 to a date and time.

DateToSec

The DateToSec instruction converts a date to the number of seconds from 00:00:00 on January 1, 1970.

Instruction	Name	FB/FUN	Graphic expression	ST expression
DateToSec	Convert Date to Seconds	FUN		Out:=DateToSec(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Date	Input	Date	Depends on data type.	Year, month, day	DT#1970-1-1
Out	Seconds	Output	Number of seconds from 00:00:00 on January 1, 1970	0 to 18446659200	Seconds	---

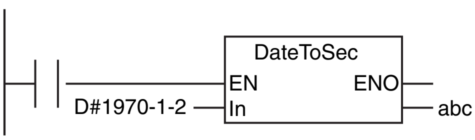
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																	OK			
Out													OK							

Function

The DateToSec instruction converts 00:00:00 on date *In* to the number of seconds from 00:00:00 on January 1, 1970. The converted value is in seconds.

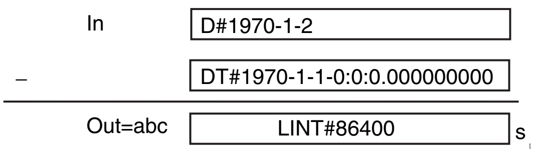
The following example is for when *In* is D#1970-1-2.

LD



ST

```
abc:=DateToSec(D#1970-1-2);
```



Additional Information

Use the SecToDate instruction (page 2-634) to convert the number of seconds from 00:00:00 on January 1,1970 to a date.

TodToSec

The TodToSec instruction converts a time of day to the number of seconds from 00:00:00.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TodToSec	Convert Time of Day to Seconds	FUN	(@)TodToSec EN ENO In Out	Out:=TodToSec(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Time of day	Input	Time of day	Depends on data type.	Hour, min-utes, seconds	TOD#0:0:0
Out	Seconds	Output	Number of seconds from 00:00:00	0 to 86399	Seconds	---

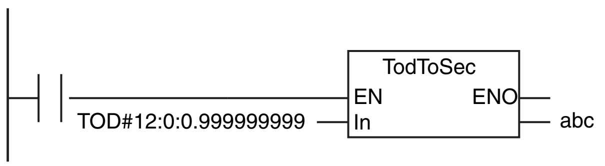
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																		OK		
Out													OK							

Function

The TodToSec instruction converts the time of day in *In* to the number of seconds from 00:00:00. The converted value is in seconds. The value is truncated below the seconds.

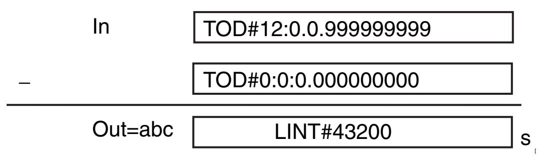
The following example is for when *In* is TOD#12:0:0.999999999.

LD



ST

abc:=TodToSec(TOD#12:0:0.999999999);

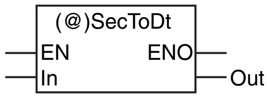


Additional Information

Use the SecToTod instruction (page 2-636) to convert the number of seconds from 00:00:00 on January 1,1970 to a time of day.

SecToDt

The SecToDt instruction converts the number of seconds from 00:00:00 on January 1, 1970 to a date and time.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SecToDt	Convert Seconds to Date and Time	FUN		Out:=SecToDt(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Seconds	Input	Number of seconds from 00:00:00 on January 1, 1970	0 to 18446744073	Seconds	0
Out	Date and time	Output	Date and time	Depends on data type.	Year, month, day, hour, minutes, seconds	---

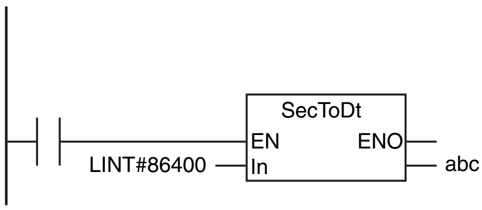
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In													OK							
Out																			OK	

Function

The SecToDt instruction converts the number of seconds from 00:00:00 on January 1, 1970 in *In* to a date and time.

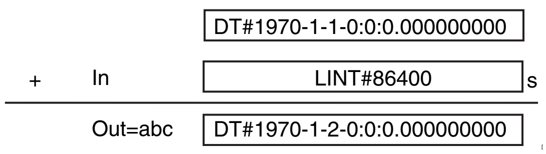
The following example is for when *In* is LINT#86400.

LD



ST

```
abc:=SecToDt(LINT#86400);
```



Related System-defined Variables

Name	Meaning	Data type	Description
_CurrentTime	System Time of Day	DT	The time of day from the system clock. The number of seconds from 00:00:00 on January 1,1970.

Additional Information

Use the DtToSec instruction (page 2-628) to convert the current time of day to the number of seconds from 00:00:00 on January 1,1970.

Precautions for Correct Use

An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.

- The value of *In* is outside of the valid range.

SecToDate

The SecToDate instruction converts the number of seconds from 00:00:00 to a date.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SecToDate	Convert Seconds to Date	FUN	(@)SecToDate EN ENO In Out	Out:=SecToDate(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Seconds	Input	Number of seconds from 00:00:00	0 to 18446744073	Seconds	0
Out	Date	Output	Date	Depends on data type.	Year, month, day	---

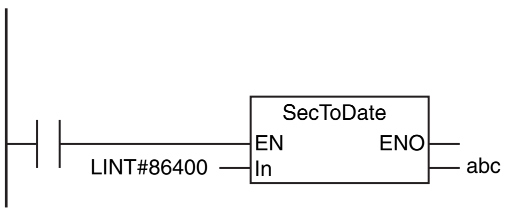
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In													OK							
Out																	OK			

Function

The SecToDate instruction converts the number of seconds from 00:00:00 in *In* to a date. The value is truncated below date.

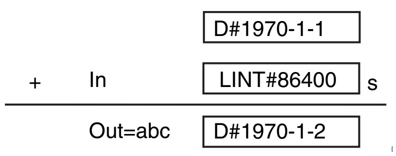
The following example is for when *In* is LINT#86400.

LD



ST

abc:=SecToDate(LINT#86400);



Additional Information

Use the DateToSec instruction (page 2-630) to convert a date to the number of seconds from 00:00:00 on January 1,1970.

Precautions for Correct Use

An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.

- The value of *In* is outside of the valid range.

SecToTod

The SecToTod instruction converts the number of seconds from 00:00:00 to a time of day.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SecToTod	Convert Seconds to Time of Day	FUN	(@)SecToTod EN ENO In Out	Out:=SecToTod(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Seconds	Input	Number of seconds from 00:00:00	Depends on data type.*	Seconds	0
Out	Time of day	Output	Time of day	Depends on data type.	Hour, minutes, seconds	---

* Negative numbers are excluded.

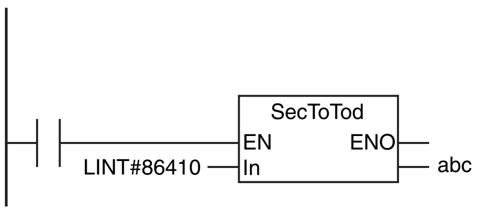
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In													OK							
Out																		OK		

Function

The SecToTod instruction converts the number of seconds from 00:00:00 in *In* to a time of day. If the value of *In* is 24 hours or longer, *In* is divided by 24 and the remainder is converted to the time of day.

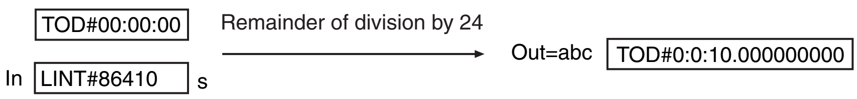
The following example is for when *In* is LINT#86410.

LD



ST

abc:=SecToTod(LINT#86410);



Additional Information

Use the TodToSec instruction (page 2-631) to convert a time of day to the number of seconds from 00:00:00 on January 1,1970.

Precautions for Correct Use

An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.

- The value of *In* is outside of the valid range.

TimeToNanoSec

The TimeToNanoSec instruction converts a time to nanoseconds.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TimeToNanoSec	Convert Time to Nanoseconds	FUN	(@)TimeToNanoSec EN ENO In Out	Out:=TimeToNanoSec(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Time	Input	Time	Depends on data type.	ns	T#0s
Out	Nanoseconds	Output	Nanoseconds	*	ns	---

* –9223372036854775808 to 9223372036854775807

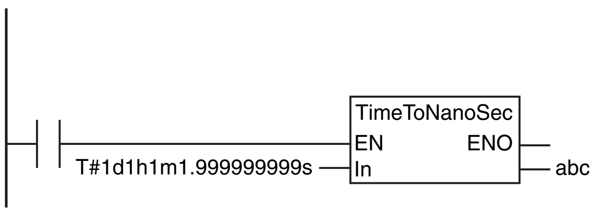
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																OK				
Out													OK							

Function

The TimeToNanoSec instruction converts the time in *In* to nanoseconds.

The following example is for when *In* is T#1d1h1m1.999999999s.

LD



ST

abc:=TimeToNanoSec(T#1d1h1m1.999999999s);

Nanoseconds

In T#1d1h1m1.999999999s → Out=abc LINT#90061000000000 ns

Additional Information

Use the NanoSecToTime instruction (page 2-640) to convert nanoseconds to a time.

TimeToSec

The TimeToSec instruction converts a time to seconds.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TimeToSec	Convert Time to Seconds	FUN	(@)TimeToSec EN ENO In Out	Out:=TimeToSec(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Time	Input	Time	Depends on data type.	ns	T#0s
Out	Seconds	Output	Seconds	–9223372036 to 9223372036	Seconds	---

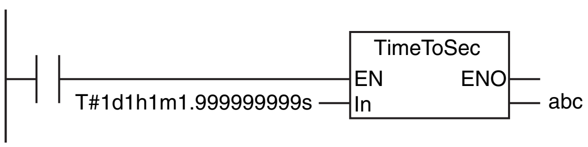
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																OK				
Out							OK													

Function

The TimeToSec instruction converts the time in *In* to seconds. The value is truncated below the seconds.

The following example is for when *In* is T#1d1h1m1.999999999s.

LD



ST

abc:=TimeToSec(T#1d1h1m1.999999999s);

In T#1d1h1m1.999999999s $\xrightarrow{\text{Seconds}}$ Out=abc LINT#90061 s

Additional Information

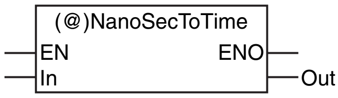
Use the SecToTime instruction (page 2-641) to convert seconds to a time.

Precautions for Correct Use

In is in nanoseconds. *Out* is in seconds.

NanoSecToTime

The NanoSecToTime instruction converts nanoseconds to a time.

Instruction	Name	FB/FUN	Graphic expression	ST expression
NanoSecToTime	Convert Nanoseconds to Time	FUN		Out:=NanoSecToTime(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Nanoseconds	Input	Nanoseconds	*	ns	0
Out	Time	Output	Time	Depends on data type.	ns	---

* –9223372036854775808 to 9223372036854775807

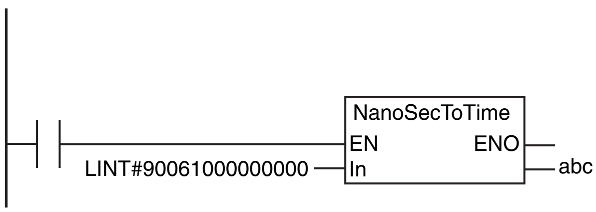
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In													OK							
Out																OK				

Function

The NanoSecToTime instruction converts the number of nanoseconds in *In* to a time.

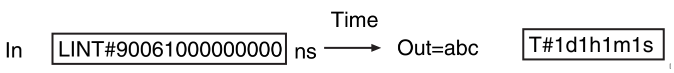
The following example is for when *In* is LINT#90061000000000.

LD



ST

abc:=NanoSecToTime(LINT#90061000000000);

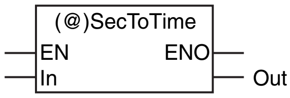


Additional Information

Use the TimeToNanoSec instruction (page 2-638) to convert a time to nanoseconds.

SecToTime

The SecToTime instruction converts seconds to a time.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SecToTime	Convert Seconds to Time	FUN		Out:=SecToTime(In);

Variables

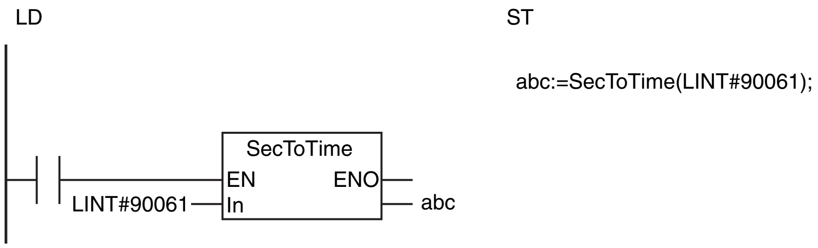
Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Seconds	Input	Seconds	–9223372036 to 9223372036	Seconds	0
Out	Time	Output	Time	Depends on data type.	ns	---

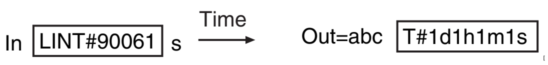
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In													OK							
Out																OK				

Function

The SecToTime instruction converts the number of seconds in *In* to a time.

The following example is for when *In* is LINT#90061.





Additional Information

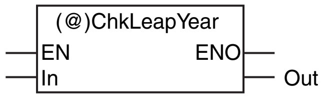
Use the TimeToSec instruction (page 2-639) to convert a time to seconds.

Precautions for Correct Use

- *In* is in seconds. *Out* is in nanoseconds.
- An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.
  - The value of *In* is outside of the valid range.

ChkLeapYear

The ChkLeapYear instruction is used to check for a leap year.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ChkLeapYear	Check for Leap Year	FUN		Out:=ChkLeapYear(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Year	Input	Year	1970 to 2554	Year	1970
Out	Result	Output	TRUE: Leap year FALSE: Not leap year	Depends on data type.	---	---

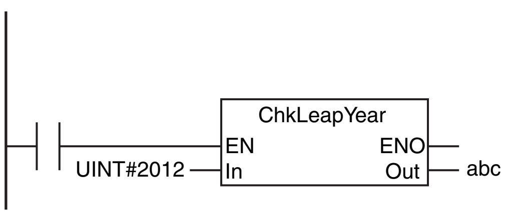
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In							OK													
Out	OK																			

Function

The ChkLeapYear instruction is used to check to see if year *In* is a leap year. If it is a leap year, the value of result *Out* is TRUE. If it is not a leap year, *Out* is FALSE.

The following example is for when *In* is UINT#2012.

LD



ST

```
abc:=ChkLeapYear(UINT#2012);
```

Precautions for Correct Use

If the value of *In* exceeds the valid range, an error will not occur and the value of *Out* will be an illegal value.

GetDaysOfMonth

The GetDaysOfMonth instruction gets the number of days in the specified month.

Instruction	Name	FB/FUN	Graphic expression	ST expression
GetDaysOfMonth	Get Days in Month	FUN	(@)GetDaysOfMonth: EN, Year, Month → ENO, Out	Out:=GetDaysOfMonth(Year, Month);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Year	Year	Input	Year	1970 to 2554	Year	1970
Month	Month		Month	1 to 12	Month	1
Out	Days	Output	Days	28 to 31	Days	---

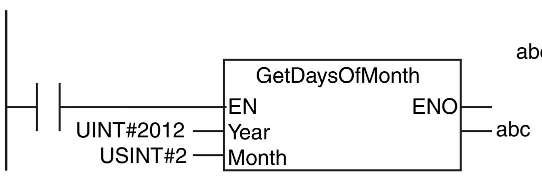
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Year							OK													
Month						OK														
Out						OK														

Function

The GetDaysOfMonth instruction gets the number of days in month *Month* of year *Year*.

The following example is for when *Year* is UINT#2012 and *Month* is USINT#2.

LD



ST

abc:=GetDaysOfMonth(UINT#2012, USINT#2);

Precautions for Correct Use

- If the value of *Year* exceeds the valid range, an error will not occur and the value of *Out* will be an illegal value.
- An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.
  - The value of *Month* is outside of the valid range.

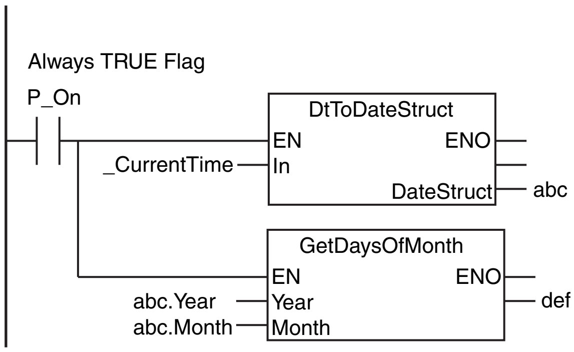
Sample Programming

This sample gets the number of days in the current month.

LD

Internal Variables	Variable	Data type	Initial value	Comment
	abc	_sDT	(Year:=0, Month:=0, Day:=0, Hour:=0, Min:=0, Sec:=0, NSec:=0)	Date and time
	def	USINT	0	Days in current month

External Variables	Variable	Data type	Constant	Comment
	_CurrentTime	DATE_AND_TIME	✔	System Time of Day



ST

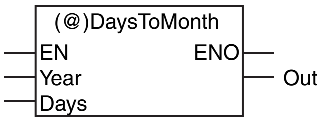
Internal Variables	Variable	Data type	Initial value	Comment
	abc	_sDT	(Year:=0, Month:=0, Day:=0, Hour:=0, Min:=0, Sec:=0, NSec:=0)	Date and time
	def	USINT	0	Days in current month

External Variables	Variable	Data type	Constant	Comment
	_CurrentTime	DATE_AND_TIME	✔	System Time of Day

```
DtToDateStruct(_CurrentTime, abc);
def:=GetDaysOfMonth(abc.Year, abc.Month);
```

DaysToMonth

The DaysToMonth instruction calculates the month based on the number of days from January 1.

Instruction	Name	FB/FUN	Graphic expression	ST expression
DaysToMonth	Convert Days to Month	FUN		Out:=DaysToMonth(Year, Days);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Year	Year	Input	Year	1970 to 2554	Year	1970
Days	Days		Number of days from January 1	1 to 365 1 to 366 for a leap year	Days	1
Out	Month	Output	Month	1 to 12	Month	---

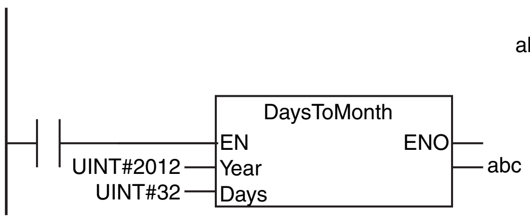
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Year							OK													
Days							OK													
Out						OK														

Function

The DaysToMonth instruction calculates the month based on the number of days in *Days* from January 1 in year *Year*.

The following example is for when *Year* is UINT#2012 and *Days* is UINT#32.

LD



ST

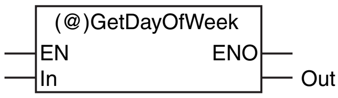
```
abc:=DaysToMonth(UINT#2012, UINT#32);
```

Precautions for Correct Use

- If the value of *Year* exceeds the valid range, an error will not occur and the value of *Out* will be an illegal value.
- An error occurs in the following case. *ENO* will be FALSE, and *Out* will not change.
  - The value of *Days* is outside of the valid range.

GetDayOfWeek

The GetDayOfWeek instruction gets the day of the week for the specified year, month, and day of month.

Instruction	Name	FB/FUN	Graphic expression	ST expression
GetDayOfWeek	Get Day of Week	FUN		Out:=GetDayOfWeek(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Year, month, day	Input	Year, month, day	Depends on data type.	Year, month, day	*
Out	Day of the week	Output	Day of the week	_MON, _TUE, _WED, _THU, _FRI, _SAT, _SUN	Day of the week	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																	OK		OK	
Out	Refer to *Function* for the enumerators for the enumerated type _eDAYOFWEEK.																			

Function

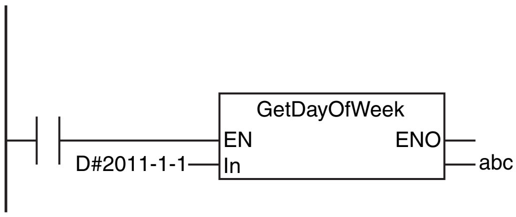
The GetDayOfWeek instruction gets the day of the week for the year, month, and day of month specified in *In*.

The data type of *Out* is enumerated type _eDAYOFWEEK. The meanings of the enumerators are as follows:

Enumerator	Meaning
_MON	Monday
_TUE	Tuesday
_WED	Wednesday
_THU	Thursday
_FRI	Friday
_SAT	Saturday
_SUN	Sunday

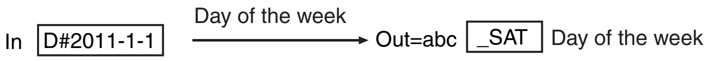
The following example is for when *In* is D#2011-1-1.

LD



ST

```
abc:=GetDayOfWeek(D#2011-1-1);
```



Related System-defined Variables

Name	Meaning	Data type	Description
_CurrentTime	System Time of Day	DT	The time of day from the system clock. The number of seconds from 00:00:00 on January 1,1970.

GetWeekOfYear

The GetWeekOfYear instruction gets the week number for the specified year, month, and day of month.

Instruction	Name	FB/FUN	Graphic expression	ST expression
GetWeekOfYear	Get Week Number	FUN	(@)GetWeekOfYear EN ENO In Out	Out:=GetWeekOfYear(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Year, month, day	Input	Year, month, day	Depends on data type.	Year, month, day	*
Out	Week	Output	Week number	1 to 54	Week	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																	OK		OK	
Out						OK														

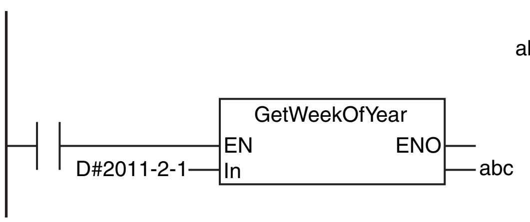
Function

The GetWeekOfYear instruction gets the week number for the year, month, and day of month specified in *In*. Weeks are counted from Monday to Sunday. The count is incremented when changing from Sunday to Monday.

January 1 is always in week 1. For example, if January 1 is a Thursday, January 1 to January 4 (Sunday) is week 1 and January 5 (Monday) to January 11 (Sunday) is week 2.

The following example is for when *In* is D#2011-2-1.

LD



ST

abc:=GetWeekOfYear(D#2011-2-1);

Week number

In D#2011-2-1 → Out=abc Week USINT#6

Related System-defined Variables

Name	Meaning	Data type	Description
_CurrentTime	System Time of Day	DT	The time of day from the system clock. The number of seconds from 00:00:00 on January 1,1970.

DtToDateStruct

The DtToDateStruct instruction converts a date and time to the year, month, day, hour, minutes, seconds, and nanoseconds.

Instruction	Name	FB/FUN	Graphic expression	ST expression
DtToDateStruct	Break Down Date and Time	FUN	(@)DtToDateStruct EN ENO In Out DateStruct	Out:=DtToDateStruct(In, DateStruct);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Date and time	Input	Date and time	Depends on data type.	Year, month, day, hour, minutes, seconds	DT#1970-1-1-0:0:0
Out	Return value	Output	Always TRUE	TRUE only	---	---
DateStruct	Date and time	Output	Date and time as a year, month, day, hour, minutes, seconds, and nanoseconds	---	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																			OK	
Out	OK																			
DateStruct	Refer to *Function* for details on the structure _sDT.																			

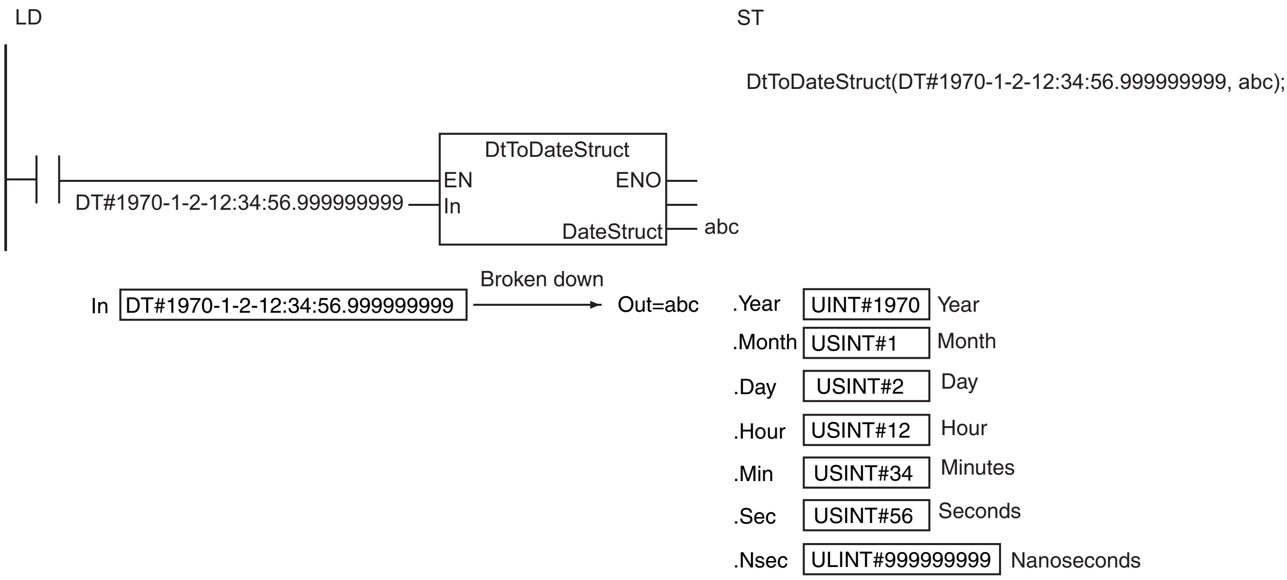
Function

The DtToDateStruct instruction converts the date and time in *In* to the year, month, day, hour, minutes, seconds, and nanoseconds. The data in the broken down date and time in *Out* is the structure _sDT. The meanings of the members are as follows:

Name	Meaning	Content	Data type	Valid range	Unit	Default
Out	Date and time	Date and time as a year, month, day, hour, minutes, seconds, and nanoseconds	_sDT	---	---	---

Name	Meaning	Content	Data type	Valid range	Unit	Default
Year	Year	Year	UINT	1970 to 2554	Year	---
Month	Month	Month	USINT	1 to 12	Month	
Day	Day	Day	USINT	1 to 31	Day	
Hour	Hour	Hour	USINT	0 to 23	Hour	
Min	Minutes	Minutes	USINT	0 to 59	Minutes	
Sec	Seconds	Seconds	USINT	0 to 59	Seconds	
Nsec	Nanoseconds	Nanoseconds	ULINT	0 to 999999999	Nanoseconds	

The following example is for when *In* is DT#1970-1-2-12:34:56.999999999.

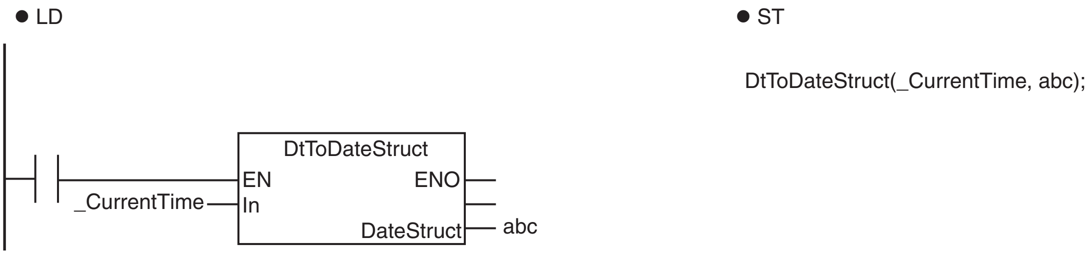


Related System-defined Variables

Name	Meaning	Data type	Description
_CurrentTime	System Time of Day	DT	The time of day from the system clock. The number of seconds from 00:00:00 on January 1,1970.

Additional Information

- Use the DateStructToDt instruction (page 2-655) to join a year, month, day, hour, minutes, seconds, and nanoseconds into a date and time.
- The following example shows how to find the current time of day.



Precautions for Correct Use

Return value *Out* is not used when the instruction is used in ST.

DateStructToDt

The DateStructToDt instruction joins a year, month, day, hour, minutes, seconds, and nanoseconds into a date and time.

Instruction	Name	FB/FUN	Graphic expression	ST expression
DateStructToDt	Join Time	FUN	(@)DateStructToDt EN ENO In Out	Out:=DateStructToDt(In);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Date and time	Input	Date and time as a year, month, day, hour, minutes, seconds, and nanoseconds	---	---	---
Out	Date and time	Output	Date and time	Depends on data type.	Year, month, day, hour, minutes, seconds	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	Refer to *Function* for details on the structure _sDT.																			
Out																			OK	

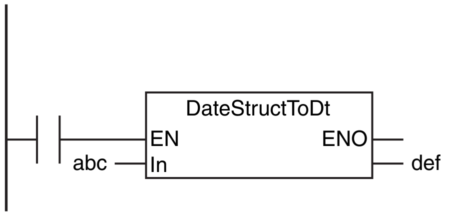
Function

The DateStructToDt instruction joins the year, month, day, hour, minutes, seconds, and nanoseconds in *In* into a date and time. The data type of *In* is structure _sDT. The meanings of the members are as follows:

Name		Meaning	Content	Data type	Valid range	Unit	Default
In		Date and time	Date and time as a year, month, day, hour, minutes, seconds, and nanoseconds	_sDT	---	---	---
	Year	Year	Year	UINT	1970 to 2554	Year	1970
	Month	Month	Month	USINT	1 to 12	Month	1
	Day	Day	Day	USINT	1 to 31	Day	
	Hour	Hour	Hour	USINT	0 to 23	Hour	0
	Min	Minutes	Minutes	USINT	0 to 59	Minutes	
	Sec	Seconds	Seconds	USINT	0 to 59	Seconds	
	Nsec	Nanoseconds	Nanoseconds	ULINT	0 to 999999999	Nanoseconds	

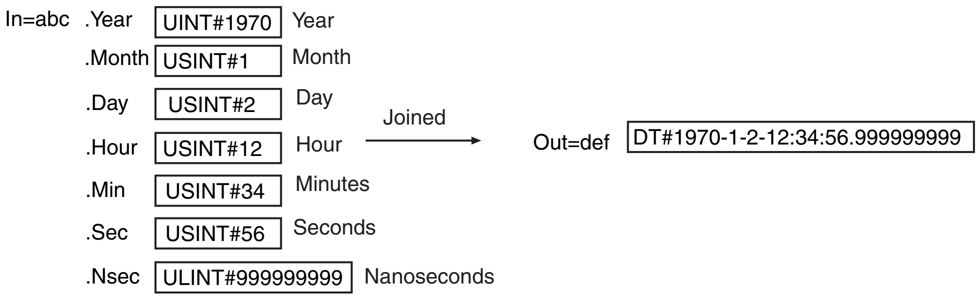
The following example is for the following values for the members of *In*: *Year* is UINT#1970, *Month* is USINT#1, *Day* is USINT#2, *Hour* is USINT#12, *Min* is USINT#34, *Sec* is USINT#56, and *Nsec* is ULINT#999999999.

LD



ST

```
def:=DateStructToDt(abc);
```



Related System-defined Variables

Name	Meaning	Data type	Description
_CurrentTime	System Time of Day	DT	The time of day from the system clock. The number of seconds from 00:00:00 on January 1,1970.

Additional Information

Use the DtToDateStruct instruction (page 2-652) to break down a date and time into a year, month, day, hour, minutes, seconds, and nanoseconds.

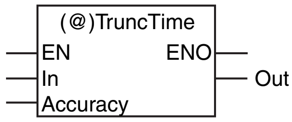
Precautions for Correct Use

An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.

- The value of a member of *In* is outside of the valid range.
- The processing result exceeds the valid range of *Out*.

TruncTime

The TruncTime instruction truncates a TIME variable below the specified time unit.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TruncTime	Truncate Time	FUN		Out:=TruncTime(In, Accuracy);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Time to truncate	Input	Time to truncate	Depends on data type.	ns	T#0s
Accuracy	Smallest unit after truncation		The smallest time unit to leave after truncation	_NANOSEC, _MICROSEC, _MILLISEC, _SEC	---	_NANO-SEC
Out	Time after truncation	Output	Time after truncation	Depends on data type.	ns	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																OK				
Accuracy	Refer to *Function* for the enumerators of enumeration type _eSUBSEC.																			
Out																OK				

Function

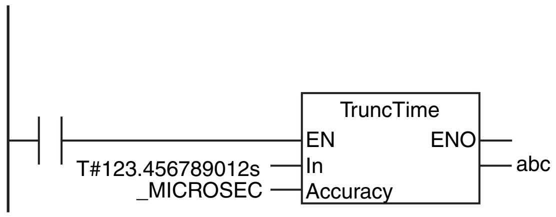
The TruncTime instruction truncates all digits below the smallest unit after truncation that is specified in *Accuracy* from the time to truncate in *In*. The resulting time after truncation is stored in time after truncation *Out*.

The data type of *Accuracy* is enumerated type _eSUBSEC. The meanings of the enumerators are as follows:

Enumerator	Meaning
_NANOSEC	Nanoseconds
_MICROSEC	Microseconds
_MILLISEC	Milliseconds
_SEC	Seconds

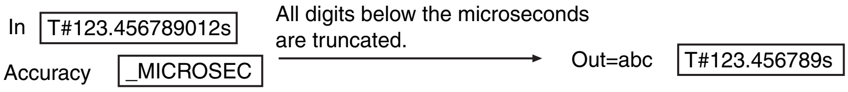
The following example is for when *In* is TIME#123.456789012s and *Accuracy* is _MICROSEC.

LD



ST

```
abc:=TruncTime(T#123.456789012s, _MICROSEC);
```



Additional Information

Before you compare two TIME variables with the EQ (=) instruction (page 2-92) or other instructions, use the TruncTime instruction to convert the two variables to the same accuracy.

Precautions for Correct Use

Version Information

A CPU Unit with unit version 1.01 or later and Sysmac Studio version 1.02 or higher are required to use this instruction.

Sample Programming

The following programming example determines if the ON time of the sensor output is equal to or greater than the threshold value. The operation mode can be either the threshold setting mode or the execution mode. The operations of these modes are described in the following table.

Operation mode	Operation
Threshold setting mode	The ON time of the sensor output is measured and the resulting value is set as the threshold.
Execution mode	The ON time of the sensor output is measured and compared with the threshold. If the ON time is equal to or greater than the threshold, the operation is considered normal.

The time is compared in milliseconds. The TruncTime instruction is used to truncate the digits in the measured time below milliseconds.
The current operation mode is stored in the *RecentMode* variable. The result is stored in the *Result* variable. The value of *Result* is TRUE if operation is normal and FALSE if there is an error.

Definitions of Global Variables

Data type: Enumeration

Variable	Enumerator	Comment
Mode		Operation mode
SET	0	Threshold setting
EXEC	1	Execution

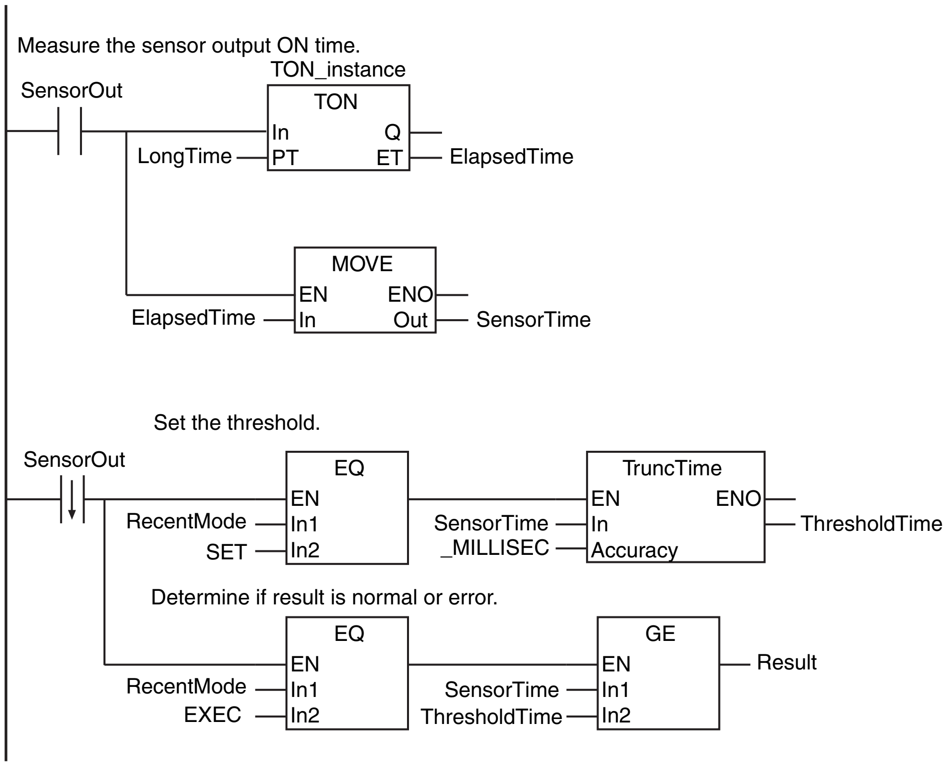
Global Variables

Variable	Data type	Initial value	Comment
RecentMode	Mode	SET	The current operation mode

LD

Internal Variables	Variable	Data type	Initial value	Comment
	SensorOut	BOOL	FALSE	Sensor output
	ElapsedTime	TIME	T#0s	Elapsed time
	SensorTime	TIME	T#0s	Sensor ON time
	LongTime	TIME	T#1h	A time that is sufficiently longer than the sensor ON time
	ThresholdTime	TIME	T#0s	Threshold
	Result	BOOL	FALSE	Result, TRUE: Normal, FALSE: Error
	TON_instance	TON		

External Variables	Variable	Data type	Comment
	RecentMode	Mode	The current operation mode



ST

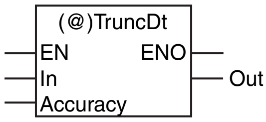
Internal Variables	Variable	Data type	Initial value	Comment
	SensorOut	BOOL	FALSE	Sensor output
	ElapsedTime	TIME	T#0s	Elapsed time
	SensorTime	TIME	T#0s	Sensor ON time
	LongTime	TIME	T#1h	A time that is sufficiently longer than the sensor ON time
	SensorDone	BOOL	FALSE	Sensor output OFF flag
	ThresholdTime	TIME	T#0s	Threshold
	Result	BOOL	FALSE	Result, TRUE: Normal, FALSE: Error
	TON_instance	TON		
	F_TRIG_instance	F_TRIG		

External Variables	Variable	Data type	Comment
	RecentMode	Mode	The current operation mode

```
// Execute TON instruction.
TON_instance(
    In:=SensorOut,    // Timer input
    PT:=LongTime,     // Set time
    ET=>ElapsedTime); // Elapsed time

// Set sensor ON time to the elapsed time of TON.
IF (SensorOut=TRUE) THEN
    SensorTime:=ElapsedTime;
END_IF;

// Detect when sensor output turns OFF.
F_TRIG_instance(Clk:=SensorOut, Q=>SensorDone);
Result:=FALSE;

// Set the threshold.
IF (SensorDone=TRUE AND RecentMode=SET) THEN
    ThresholdTime:=TruncTime(
                        In      :=SensorTime,
                        Accuracy:=_MILLISEC);  // Accuracy is milliseconds.
// Determine if result is normal or error.
ELSIF (SensorDone=TRUE AND RecentMode=EXEC) THEN
    IF (SensorTime >= ThresholdTime) THEN
        Result:=TRUE;
    END_IF;
END_IF;
```

TruncDt

The TruncDt instruction truncates a DT variable below the specified time unit.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TruncDt	Truncate Date and Time	FUN		Out:=TruncDt(In, Accuracy);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Date and time to truncate	Input	Date and time to truncate	Depends on data type.	Year, month, day, hour, minutes, seconds	DT#1970-1-1-0:0:0
Accuracy	Smallest unit after truncation		The smallest time unit to leave after truncation	_NANOSEC, _MICROSEC, _MILLISEC, _SEC	---	_NANOSEC
Out	Date and time after truncation	Output	Date and time after truncation	Depends on data type.	Year, month, day, hour, minutes, seconds	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																			OK	
Accuracy	Refer to *Function* for the enumerators of enumeration type _eSUBSEC.																			
Out																			OK	

Function

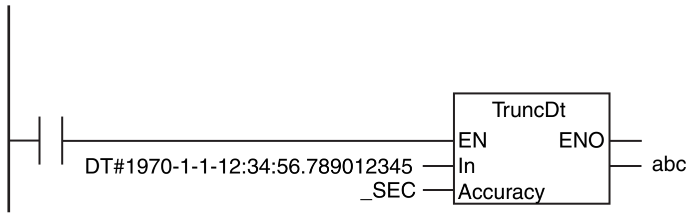
The TruncDt instruction truncates all digits below the smallest unit after truncation that is specified in *Accuracy* from the date and time to truncate in *In*. The resulting date and time after truncation is stored in date and time after truncation *Out*.

The data type of *Accuracy* is enumerated type _eSUBSEC. The meanings of the enumerators are as follows:

Enumerator	Meaning
_NANOSEC	Nanoseconds
_MICROSEC	Microseconds
_MILLISEC	Milliseconds
_SEC	Seconds

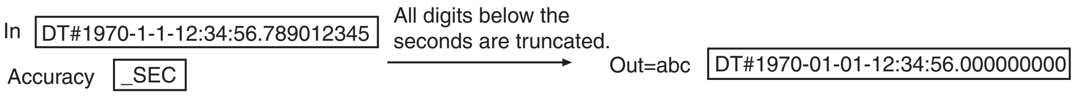
The following example is for when *In* is DT#1970-1-1-12:34:56.789012345 and *Accuracy* is _SEC.

LD



ST

```
abc:=TruncDt(DT#1970-1-1-12:34:56.789012345,
             _SEC);
```



Additional Information

Before you compare two DT variables with the EQ (=) instruction (page 2-92) or other instructions, use the TruncDt instruction to convert the two variables to the same accuracy.

Precautions for Correct Use

Version Information

A CPU Unit with unit version 1.01 or later and Sysmac Studio version 1.02 or higher are required to use this instruction.

Sample Programming

The following programming example records the date and time and the current voltage when a sensor output turns ON.

The date and time is recorded in milliseconds.

The sensor output is stored in *SensorOut* and the voltage is stored in *Voltage*. The current date and time is obtained with the GetTime instruction.

The date and times and the voltages are stored in order in a *Stack* variable as *Recent* structures whose members are the date and time and corresponding voltage.

Definitions of Global Variables

Data Types

Variable	Data type	Comment
Record	STRUCT	Structure
DandT	DT	Date and time
Voltage	REAL	Voltage

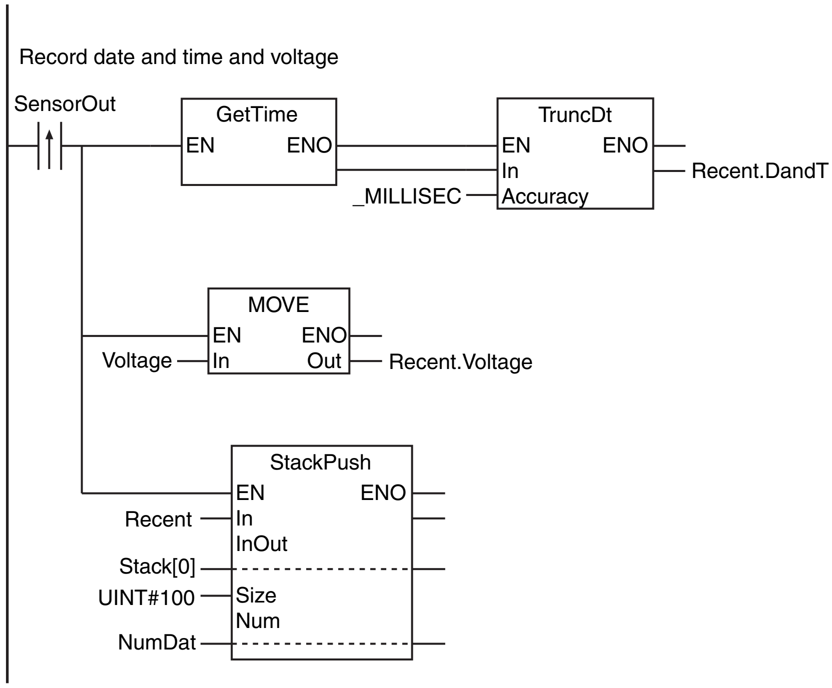
Global Variables

Variable	Data type	Initial value	Comment
Recent	Record	(DandT:=DT#1970-1-1-0:0:0,Voltage:=0.0)	Present value
Stack	ARRAY[0..99] OF Record	[100((DandT:=DT#1970-1-1-0:0:0,Voltage:=0.0))]	Stack

LD

Internal Variables	Variable	Data type	Initial value	Comment
	SensorOut	BOOL	FALSE	Sensor output
	Voltage	REAL	0.0	Voltage
	NumDat	UINT	UINT#0	Current number of stored data

External Variables	Variable	Data type	Comment
	Recent	Record	Present value
	Stack	ARRAY[0..99] OF Record	Stack



ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	FALSE	Trigger
	SensorOut	BOOL	FALSE	Sensor output
	Voltage	REAL	0.0	Voltage
	NumDat	UINT	UINT#0	Current number of stored data
	R_TRIG_instance	R_TRIG		

External Variables	Variable	Data type	Comment
	Recent	Record	Present value
	Stack	ARRAY[0..99] OF Record	Stack

```
// Activate trigger when sensor output turns ON.
R_TRIG_instance(SensorOut, Trigger);

IF (Trigger=TRUE) THEN
    // Store the current date and time down to the milliseconds.
    Recent.DandT:=TruncDt(
                  In      :=GetTime(),  // Get the date and time.
                  Accuracy:=_MILLISEC); // Accuracy is milliseconds.

    // Get current voltage.
    Recent.Voltage:=Voltage;

    // Record date and time and voltage in stack.
    StackPush(
        In   :=Recent,   // Date and time, and voltage
        InOut:=Stack[0], // Stack array
        Size :=UINT#100, // Number of stack array elements: 100
        Num  :=NumDat);  // Number of data currently stored
END_IF;
```

TruncTod

The TruncTod instruction truncates a TOD variable below the specified time unit.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TruncTod	Truncate Time of Day	FUN	(@)TruncTod EN ENO In Out Accuracy	Out:=TruncTod(In, Accuracy);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Time of day to truncate	Input	Time of day to truncate	Depends on data type.	Hour, minutes, seconds	TOD#0:0:0
Accuracy	Smallest unit after truncation		The smallest time unit to leave after truncation	_NANOSEC, _MICROSEC, _MILLISEC, _SEC	---	_NANOSEC
Out	Time of day after truncation	Output	Time of day after truncation	Depends on data type.	Hour, minutes, seconds	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																		OK		
Accuracy	Refer to *Function* for the enumerators of enumeration type _eSUBSEC.																			
Out																		OK		

Function

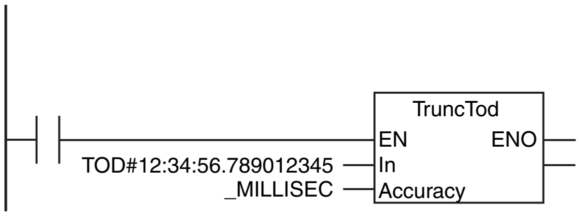
The TruncTod instruction truncates all digits below the smallest unit after truncation that is specified in *Accuracy* from the time of day to truncate in *In*. The resulting time of day after truncation is stored in time of day after truncation *Out*.

The data type of *Accuracy* is enumerated type _eSUBSEC. The meanings of the enumerators are as follows:

Enumerator	Meaning
_NANOSEC	Nanoseconds
_MICROSEC	Microseconds
_MILLISEC	Milliseconds
_SEC	Seconds

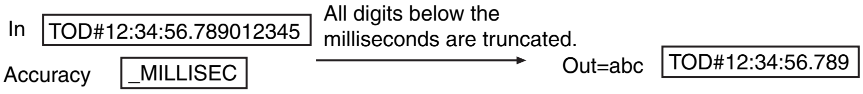
The following example is for when *In* is TOD#12:34:56.789012345 and *Accuracy* is _MILLISEC.

LD



ST

```
abc:=TruncTod(TOD#12:34:56.789012345,
              _MILLISEC);
```



Additional Information

Before you compare two TOD variables with the EQ (=) instruction (page 2-92) or other instructions, use the TruncTod instruction to convert the two variables to the same accuracy.

Precautions for Correct Use

Version Information

A CPU Unit with unit version 1.01 or later and Sysmac Studio version 1.02 or higher are required to use this instruction.

Sample Programming

The following programming example records the time of day and the current voltage when a sensor output turns ON.

The time of day is recorded in seconds.

The sensor output is stored in *SensorOut* and the voltage is stored in *Voltage*. The current time of day is obtained with the GetTime and DT_TO_TOD instructions.

The times of day and the voltages are stored in order in a *Stack* variable as *Recent* structures whose members are the time of day and corresponding voltage.

Definitions of Global Variables

Data Types

Variable	Data type	Comment
Record	STRUCT	Structure
TofD	TOD	Time of day
Voltage	REAL	Voltage

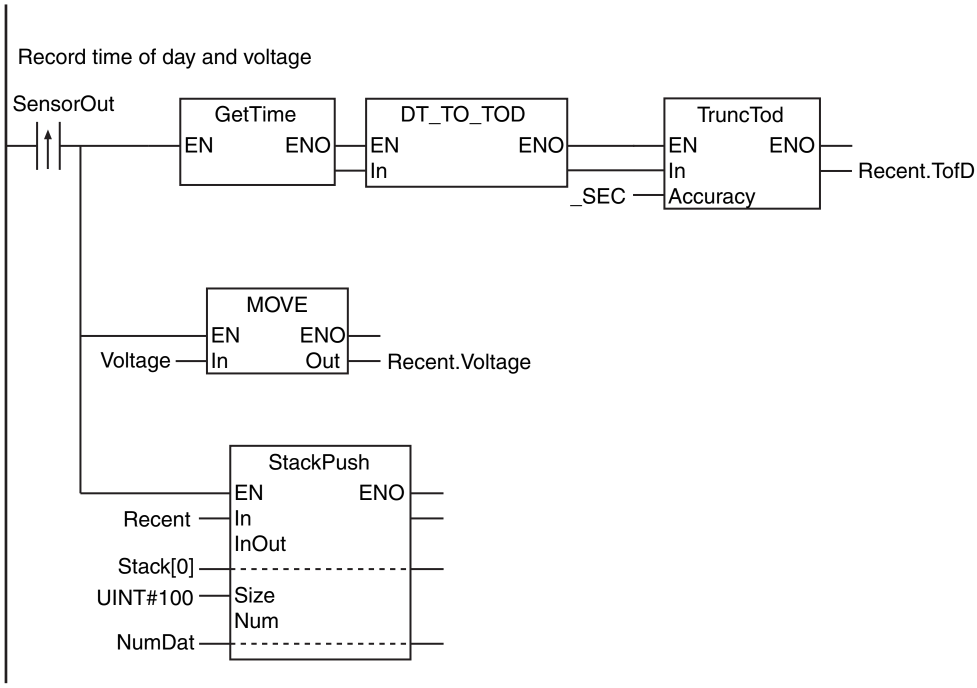
Global Variables

Variable	Data type	Initial value	Comment
Recent	Record	(TofD:=TOD#0:0:0,Voltage:=0.0)	Present value
Stack	ARRAY[0..99] OF Record	[100((TofD:=TOD#0:0:0,Voltage:=0.0))]	Stack

LD

Internal Variables	Variable	Data type	Initial value	Comment
	SensorOut	BOOL	FALSE	Sensor output
	Voltage	REAL	0.0	Voltage
	NumDat	UINT	UINT#0	Current number of stored data

External Variables	Variable	Data type	Comment
	Recent	Record	Present value
	Stack	ARRAY[0..99] OF Record	Stack



ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	FALSE	Trigger
	SensorOut	BOOL	FALSE	Sensor output
	TmpTod	TOD	TOD#0:0:0	Temporary variable
	Voltage	REAL	0.0	Voltage
	NumDat	UINT	UINT#0	Current number of stored data
	R_TRIG_instance	R_TRIG		

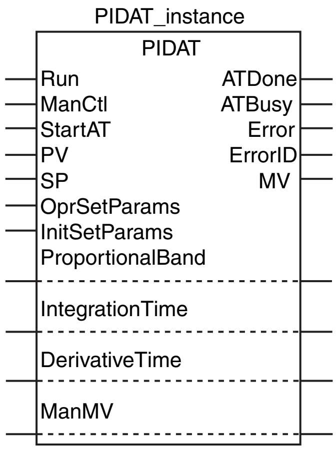
External Variables	Variable	Data type	Comment
	Recent	Record	Present value
	Stack	ARRAY[0..99] OF Record	Stack

```
// Activate trigger when sensor output turns ON.
R_TRIG_instance(SensorOut, Trigger);

IF (Trigger=TRUE) THEN
    // Store the current time of day down to the seconds.
    TmpTod     :=DT_TO_TOD(GetTime()); // Get time of day.
    Recent.TofD:=TruncTod(
                     In      :=TmpTod,
                     Accuracy:=_SEC);   // Accuracy is seconds.

    // Get current voltage.
    Recent.Voltage:=Voltage;

    // Record time of day and voltage in stack.
    StackPush(
        In   :=Recent,   // Time of day and voltage
        InOut:=Stack[0], // Stack array
        Size :=UINT#100, // Number of stack array elements: 100
        Num  :=NumDat);  // Number of data currently stored
END_IF;
```

Analog Control Instructions

PIDAT

The PIDAT instruction performs PID control with autotuning (2-PID control with set point filter).

Instruction	Name	FB/FUN	Graphic expression	ST expression
PIDAT	PID Control with Autotuning	FB		PIDAT_instance(Run, ManCtl, StartAT, PV, SP, OprSetParams, InitSetParams, ProportionalBand, IntegrationTime, DerivativeTime, ManMV, ATDone, ATBusy, Error, ErrorID, MV);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Run	Execution condition	Input	TRUE: Execute FALSE: Stop	Depends on data type.	---	FALSE
ManCtl	Manual/auto control		TRUE: Manual operation FALSE: Automatic operation			
StartAT	Autotuning execution condition		TRUE: Execute FALSE: Cancel			
PV	Process value		Process value	*1		
SP	Set point		Set point			
OprSet Params	Operation setting parameters		Parameters set during operation	---		0
InitSet Params	Initial setting parameters		Initial setting parameters			---
Proportional Band	Proportional band	In-out	Proportional band	0.01 to 1000.00	% FS	---
Integration-Time	Integration time		Integration time The higher the value is, the weaker the integral action is. No integral action is performed for 0.	T#0.0000s to T#10000.0000s*2	s	
DerivativeTime	Derivative time		Derivative time The higher the value is, the stronger the derivative action is. No derivative action is performed for 0.	T#0.0000s to T#10000.0000s*2		
ManMV	Manual manipulated variable		Manual manipulated variable	–320 to 320	%	

Name	Meaning	I/O	Description	Valid range	Unit	Default
ATDone	Autotuning normal completion	Output	TRUE: Normal completion FALSE: *3	Depends on data type.	---	---
ATBusy	Autotuning busy		TRUE: Autotuning FALSE: Not autotuning			
MV	Manipulated variable		Manipulated variable	–320 to 320	%	

*1 Value of input range lower limit *InitSetParams.RngLowLmt* to Value of input range upper limit *InitSetParams.RngUpLmt*

*2 Digits below 0.0001 s are truncated.

*3 FALSE indicates an error end, that PID control is in progress without autotuning, or that PID control is not in progress.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Run	OK																			
ManCtl	OK																			
StartAT	OK																			
PV														OK						
SP														OK						
OprSetParams	Refer to *Function* for details on the structure _sOPR_SET_PARAMS.																			
InitSetParams	Refer to *Function* for details on the structure _sINIT_SET_PARAMS.																			
ProportionalBand														OK						
IntegrationTime																OK				
DerivativeTime																OK				
ManMV														OK						
ATDone	OK																			
ATBusy	OK																			
MV														OK						

Function

The PIDAT instruction performs PID control of a manipulated variable for a temperature controller or other device. PID control is started when the value of execution condition *Run* changes to TRUE. While the value of *Run* is TRUE, the following process is repeated periodically: process value *PV* is read, PID processing is performed, and manipulated variable *MV* is output. PID control is stopped when the value of *Run* changes to FALSE.

Autotuning is supported to automatically find the optimum PID constants. When the value of the autotuning execution condition *StartAT* changes to TRUE, the PID constants are autotuned.

Structure Specifications

The data type of operation setting parameter *OprSetParams* is structure _sOPR_SET_PARAMS. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
OprSetParams	Operation Setting Parameters	Parameters that are set during operation.	_sOPR_SET_PARAMS	---	---	---
MVLowLmt	MV Lower Limit	The lower limit of the MV.	REAL	–320 to 320*	%	0
MVUpLmt	MV Upper Limit	The upper limit of the MV.	REAL			100
ManResetVal	Manual Reset Value	The value of *MV* when the deviation is 0 for the proportional action.	REAL	–320 to 320		0
MVTrackSw	MV Tracking Switch	TRUE: ON FALSE: OFF	BOOL	Depends on data type.	---	FALSE
MVTrackVal	MV Tracking Value	The value that is set in *MV* during MV tracking.	REAL	–320 to 320	%	0
StopMV	Stop MV	The value that is set in *MV* when instruction execution is stopped.	REAL			
ErrorMV	Error MV	The value that is set in *MV* when an error occurs.	REAL			
Alpha	2-PID Parameter α	The set point filter is disabled if the set point filter coefficient α is 0.	REAL	0.00 to 1.00	---	0.65
ATCalcGain	Autotuning Calculation Gain	Adjustment coefficient from autotuning results. Stability is given higher priority with higher values. The speed of response is given higher priority with lower values.	REAL	0.1 to 10.0		1.0
ATHystrs	Autotuning Hysteresis	The hysteresis of the limit cycle.	REAL		% FS	0.2

* *MVLowLmt* must be less than *MVUpLmt*.

The data type of initial setting parameter *InitSetParams* is structure _sINIT_SET_PARAMS. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
InitSetParams	Initial Setting Parameters	Initial setting parameters.	_sINIT_SET_PARAMS	---	---	---
SampTime	Sampling Period	The period for PID processing.	TIME	T#0.0001s to #100.0000s	s	T#0.1s
RngLowLmt	Lower Limit of Input Range	The lower limit of *PV* and *SP*.	REAL	–32000 to 32000*	---	0
RngUpLmt	Upper Limit of Input Range	The upper limit of *PV* and *SP*.	REAL			100
DirOpr	Action Direction	TRUE: Forward action FALSE: Reverse action	BOOL	Depends on data type.		FALSE

* *RngLowLmt* must be less than *RngUpLmt*.

Meanings of Variables

The meanings of the variables that are used in this command are described below.

● *Run* (Execution Condition)

This is the execution condition for the instruction. PID control is performed while the value is TRUE. PID control is stopped when the value changes to FALSE.

● *ManCtl* (Manual/Auto Control)

This instruction can be executed in one of two modes: Manual operation or automatic operation. The value of *ManCtl* determines which mode is used.

Value of *ManCtl*	Operation mode	Value of *MV*
TRUE	Manual	Value of *ManMV* (PID control is not performed.)
FALSE	Automatic	Value that is calculated for PID control

● *StartAT* (Autotuning Execution Condition)

This is the execution condition for autotuning the PID constants. If the value of *StartAT* is TRUE when the value of *Run* changes to TRUE, autotuning is performed when PID control is started. If the value of *StartAT* changes to TRUE during PID control (i.e., when the value of *Run* is TRUE), autotuning is performed during PID control. In either case, autotuning is canceled if the value of *StartAT* changes to FALSE during autotuning. Autotuning is described in more detail later.

● *PV* (Process Value)

This is the process value of the controlled system.

● *SP* (Set Point)

This is the set point for the controlled system.

● *MVLowLmt* (MV Lower Limit) and *MVUpLmt* (MV Upper Limit)

You can limit the value of *MV*. *MVLowLmt* and *MVUpLmt* are the lower and upper limits to *MV*. *MVLowLmt* must always be less than *MVUpLmt*.

MV from PID processing	Value of *MV*
Less than *MVLowLmt*	MVLowLmt
Between *MVLowLmt* and *MVUpLmt*, inclusive	*MV* from PID processing
Greater than *MVUpLmt*	MVUpLmt

If stop MV *StopMV*, error MV *ErrorMV*, or manual MV *ManMV* is set in manipulated variable *MV*, limit control is not applied.

You can change *MVLowLmt* or *MVUpLmt* even if the control status of this instruction is not autotuning during automatic operation.

However, if you change *MVLowLmt* or *MVUpLmt* to an expansion direction during operation, the value of *MV* which is the same as one in the last sampling period is output and changed smoothly at this time (bumpless).

Repeated changing of *MVLowLmt* or *MVUpLmt* will effect the control performance so that sufficient control performance may not obtain.

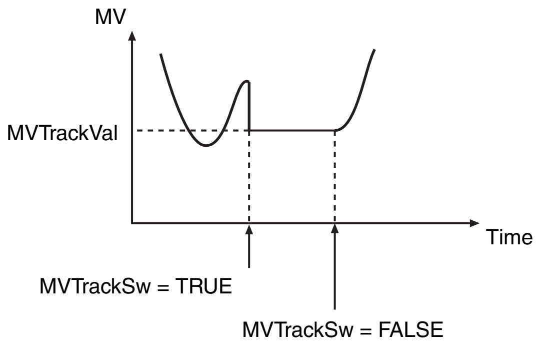
Confirm the effects on the control performance before you repeatedly change *MVLowLmt* or *MVUpLmt* during operation.

- ***ManResetVal* (Manual Reset Value)**

This is the value of *MV* when the deviation (i.e., the difference between *PV* and *SP*) is 0 for the proportional action. The value of *ManResetVal* determines the location of the proportional action band. When integral operation is performed, the manual reset value is ignored. Therefore, the setting of *ManResetVal* is enabled when the value of *IntegrationTime* is 0.

- ***MVTrackSw* (MV Tracking Switch)**

MV tracking is a function that sets the *MV* to an external input value (called the MV tracking value) during automatic operation. MV tracking is performed while the value of *MVTrackSw* is TRUE. When the value of *MVTrackSw* changes to FALSE, the value of MV returns to the result of PID processing. The value of MV is changed smoothly at this time (bumpless).



- ***MVTrackVal* (MV Tracking Value)**

This is the value to which *MV* is set during MV tracking. The value of *MVTrackVal* is limited by the values of *MVLowLmt* and *MVUpLmt*.

- ***StopMV* (Stop MV)**

This is the value to which *MV* is set when the value of *Run* is FALSE (i.e., when execution of this instruction is stopped).

- ***ErrorMV* (Error MV)**

This is the value to which *MV* is set when an error occurs (i.e., when the value of *Error* is TRUE). If the value of *ErrorMV* is not within the valid range (–320 to 320), the value of *MV* will be 0 when an error occurs.

- ***Alpha* (2-PID Parameter α)**

This parameter determines the coefficient of the set point filter. Refer to the description in *2-PID Control with Set Point Filter* on page 2-680 for details. Normally set the value of *Alpha* to 0.65.

- ***ATCalcGain* (Autotuning Calculation Gain)**

This variable gives the coefficient of the PID constants that were calculated by autotuning when they are applied to the actual PID constants. If a value of 1.00 is specified, the results of autotuning are used directly. Increase the value of *ATCalcGain* to give priority to stability and decrease it to give priority to response.

- ***ATHystrs* (Autotuning Hysteresis)**

This is the hysteresis that is used in the limit cycle for autotuning. More accurate tuning is achieved if the value of *ATHystrs* is small. However, if the process value is not stable and proper autotuning is difficult, increase the value. Refer to the description of autotuning for details.

- ***SampTime* (Sampling Period)**

This is the minimum value of the period for PID processing. Refer to the description of the execution timing of PID processing for details. PID processing is not performed again until the time specified for *SampTime* has elapsed since the last time PID processing was performed.

- ***RngLowLmt* (Lower Limit of Input Range) and *RngUpLmt* (Upper Limit of Input Range)**

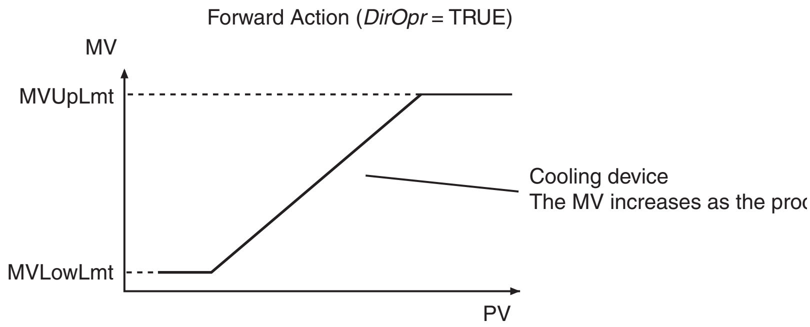
These are the lower limit and upper limit of *PV* and *SP*. An error will occur if the value of the parameter connected to *PV* or *SP* exceeds either of these limits. *RngLowLmt* must always be less than *RngUpLmt*.

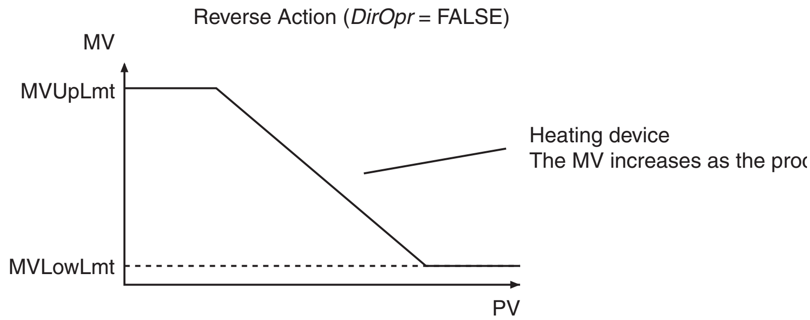
- ***DirOpr* (Action Direction)**

This variable specifies if *MV* is increased or decreased for changes in the value of *PV*. These are called a forward action and a reverse action.

Value of *DirOpr*	Meaning	Value of *MV*
TRUE	Forward action	Increases with the value of *PV*.
FALSE	Reverse action	Decreases with the value of *PV*.

The difference between a forward action and reverse action are described here for temperature control. A forward action is used to control the MV for a cooling device. That is, the higher the process temperature, the larger the MV of the cooling device must be. On the other hand, a reverse action is used to control the MV for a heating device. That is, the lower the process temperature, the larger the MV of the heating device must be.





- ***ProportionalBand* (Proportional Band)**

This is one of the three PID constants. Refer to the description of the proportional action for details. The larger the *ProportionalBand* is, the greater the offset is. Hunting occurs if the *ProportionalBand* is too small.

- ***IntegrationTime* (Integration Time)**

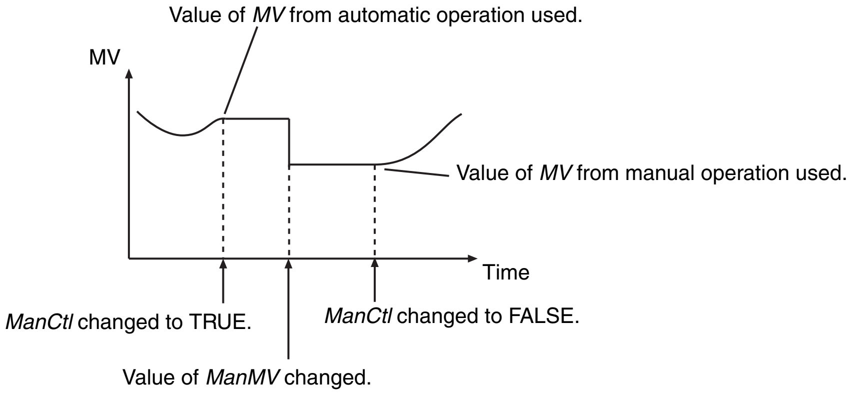
This is one of the three PID constants. Refer to the description of the integral action for details. The larger the value of *IntegrationTime* is, the weaker the integral action is.

DerivativeTime (Derivative Time)

This is one of the three PID constants. Refer to the description of the derivative action for details. The larger the value of *DerivativeTime* is, the stronger the derivative action is.

ManMV (Manual Manipulated Variable)

MV is set to this value during manual operation (while *ManCtl* is TRUE). However, immediately after changing from automatic to manual operation, the value of MV from automatic operation is used. *MV* is set to the value of *ManMV* only when it changes after operation switches to manual operation. When operation changes from manual to automatic operation, the value of MV from manual operation is used. The value of *ManMV* does not have to be between *MVLowLmt* and *MVUpLmt*.



ATDone (Autotuning Normal Completion)

This flag indicates when autotuning was completed normally. It changes to TRUE when autotuning is completed normally and remains TRUE as long as the value of *StartAT* is TRUE. It is FALSE in the following cases.

- An autotuning error end occurred.
- Autotuning is in progress (i.e., while the value of *ATBusy* is TRUE).
- PID control is in progress without autotuning.
- PID control is not in progress (i.e., the value of *Run* is FALSE).
- The value of *StartAT* is FALSE.

ATBusy (Autotuning Busy)

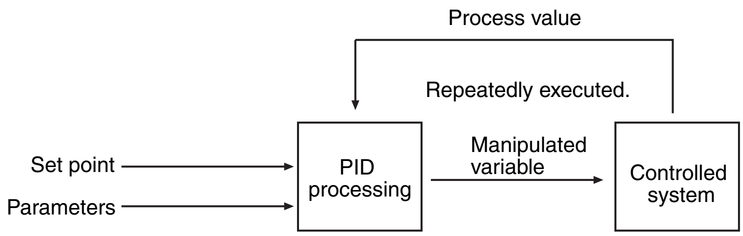
This flag indicates when autotuning is in progress. It is TRUE while autotuning is in progress. Otherwise it is FALSE.

MV (Manipulated Variable)

This is the manipulated variable that is applied to the controlled system.

Introduction to PID Control

PID control is a feedback control method that repeatedly measures the process value of the controlled system and calculates a manipulated variable so that the process value approaches a set point. This instruction therefore outputs a manipulated variable for the following inputs: process value, set point, and calculation parameters. PID control periodically measures the process value, calculates the manipulated variable, and outputs the manipulated variable so that the process value approaches the set point.

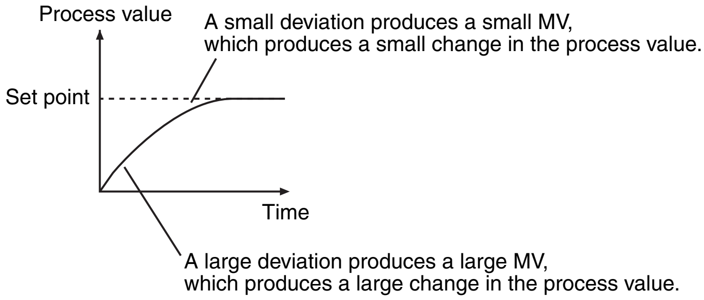


Proportional (P), Integral (I), and Derivative (D) Actions

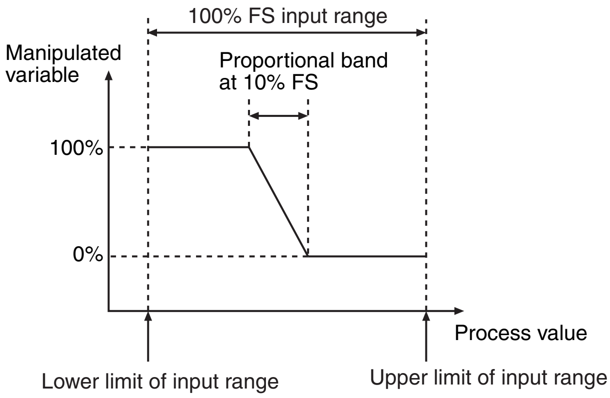
PID control is performed by combining the proportional action, integral action, and derivative action. These actions are described next.

● Proportional Action (P)

The proportional action increases the absolute value of the manipulated variable in proportion to the deviation between the process value and the set point. The process value of the controlled system changes as shown below.

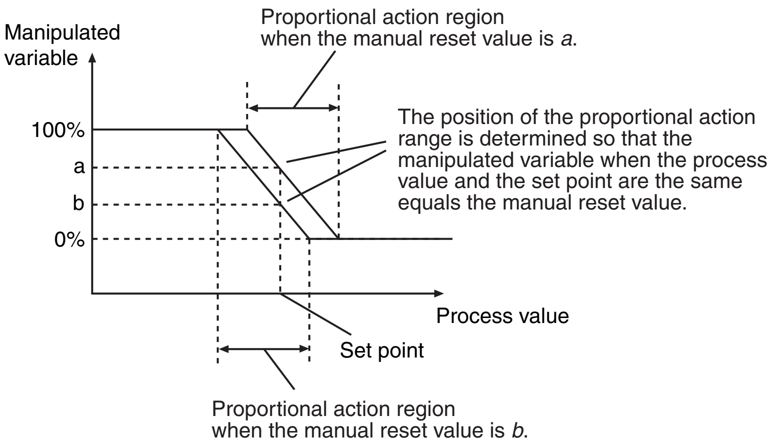


The proportional band is one of the settings that are used for the proportional action. The proportional band is the range of the process value to which the proportional action is applied. If the process value is not in the proportional band, the manipulated variable is set to 100% or 0%.
The proportional band is expressed as the percentage of the input range in which to perform the proportional action (% FS). The following diagram shows the proportional band set to 10% FS.

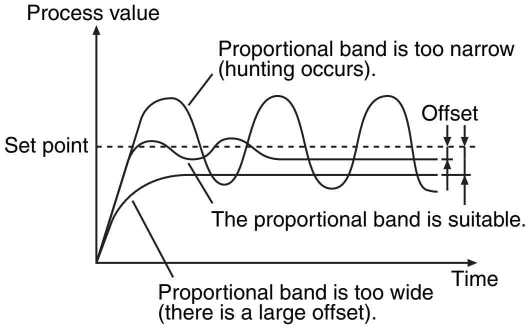


Another parameter for the proportional action is the manual reset value. The manual reset value is the manipulated variable that is used when the deviation is 0. The manual reset value determines the position of the proportional action range in the process value–manipulated variable graph. The relationship between the manual reset value and the proportional action region is shown below.

The position of the proportional action range is determined so that the manipulated variable when the process value and the set point are the same equals the manual reset value.



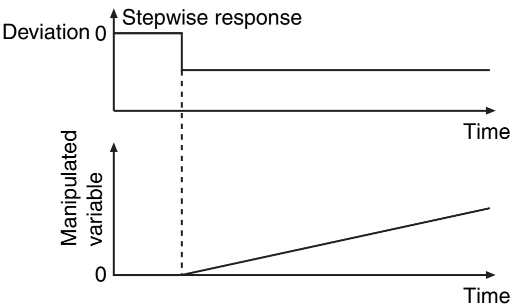
If the manual reset value is not suitable, the deviation will never reach 0. The remaining deviation is called the offset or the residual deviation. You can make the proportional band narrower to reduce the offset. If the proportional band is too narrow, the process value will not stop at the set point. This is called overshooting. If the process value does not stabilize and oscillates around the set point, it is called hunting.



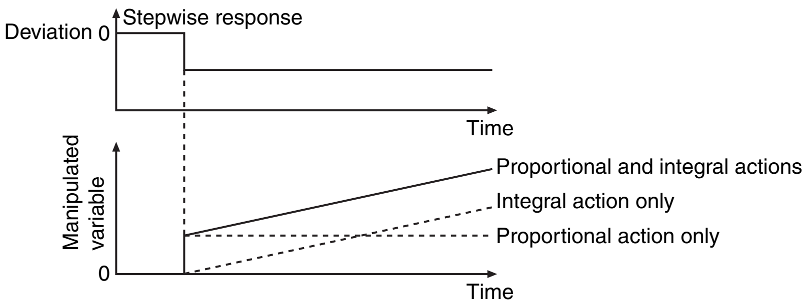
● Integral Action (I)

Very accurate adjustment of the proportional band and manual reset value is required to bring the offset to 0 with only the proportional action. Also, the size of the offset varies with the disturbance, so it is necessary to repeat the adjustment frequently. To simplify the operation, an integral action is used in combination with the proportional action. The integral action integrates the deviation on the time axis and then increases the absolute value of the manipulated variable in proportion to the result. When normal distribution operation is performed, the manual reset value is ignored. The following graph on the left shows changes in the manipulated variable for the integral action when a deviation occurs in stepwise fashion. The following graph on the right shows changes in the manipulated variable when the integral and proportional actions are combined.

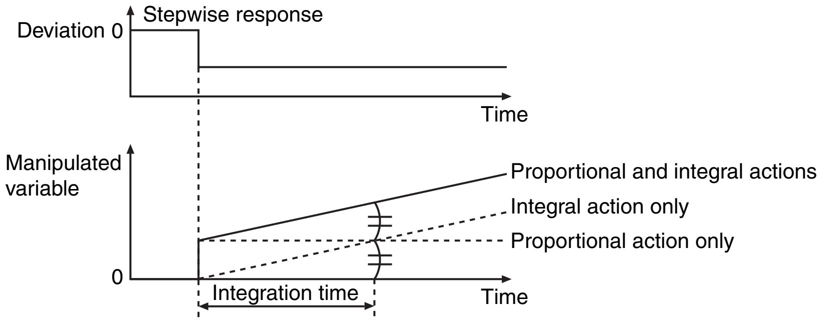
Manipulated Variable for Integral Action



Manipulated Variable for Integral and Proportional Actions Together

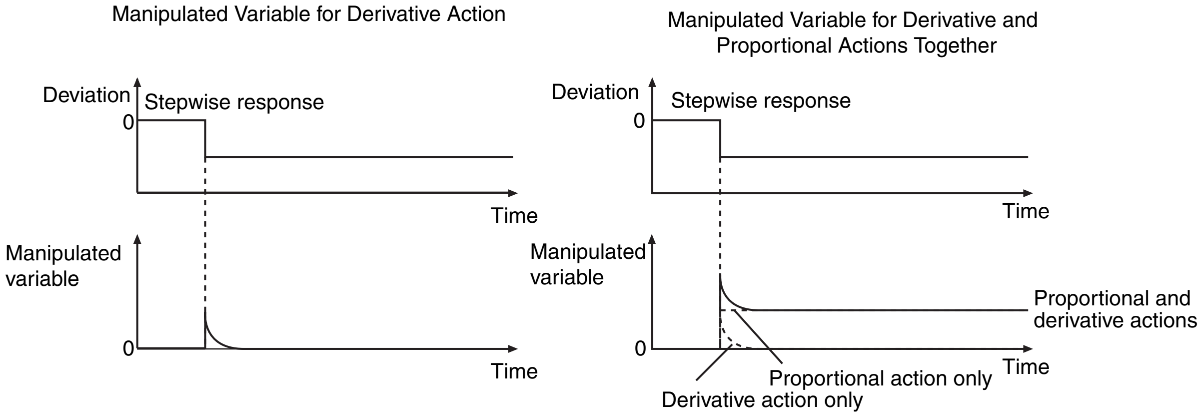


One of the parameters for the integral action is the integration time. This is the time for the manipulated variable from the integral action to equal the manipulated variable from the proportional action when a stepwise deviation occurs. The shorter the integration time is, the stronger the integral action is. A short integration time reduces the time for the offset to reach 0, but it can also cause hunting.

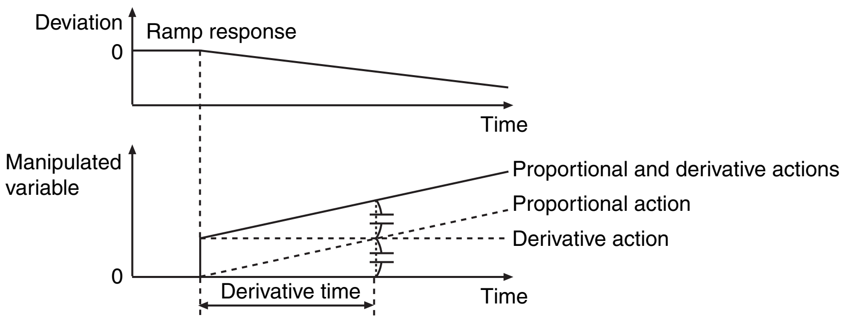


● Derivative Action (D)

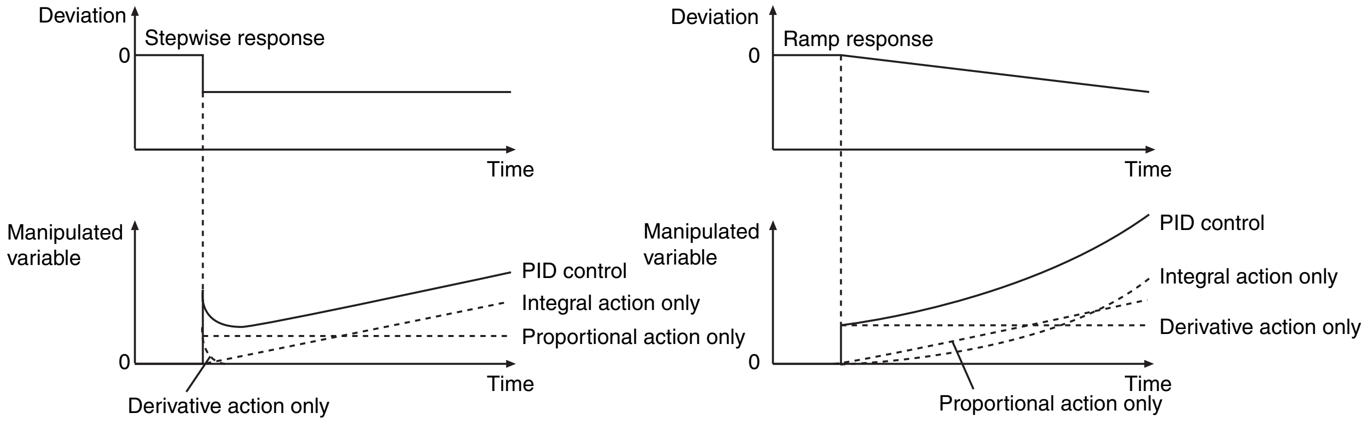
If the proportional and integral actions are used together, the offset will reach 0 and the process value will reach the set point. However, if disturbance causes the process value to change quickly, time is required to restore the original state. The derivative action functions to quickly return the process value to the set point when there is a disturbance. The derivative action differentiates the deviation on the time axis and then increases the absolute value of the manipulated variable in proportion to the result. In other words, the larger the change in the process value is, the larger the absolute value of the manipulated variable for the derivative action is. The changes in the manipulated variable for the derivative action when a deviation occurs in stepwise fashion are shown below. The changes in the manipulated variable when the derivative and proportional actions are combined are also shown.



One of the parameters for the derivative action is the derivative time. This is the time for the manipulated variable from the derivative action to equal the manipulated variable from the proportional action when a ramp deviation occurs. The longer the derivative time is, the stronger the derivative action is. A long derivative time provides a rapid response to disturbances, but it can also cause hunting.



The total of the manipulated variables for the proportional, integral, and derivative actions is the manipulated variable for PID control. The changes in the manipulated variable for PID control for a stepwise and ramp deviations are shown below.

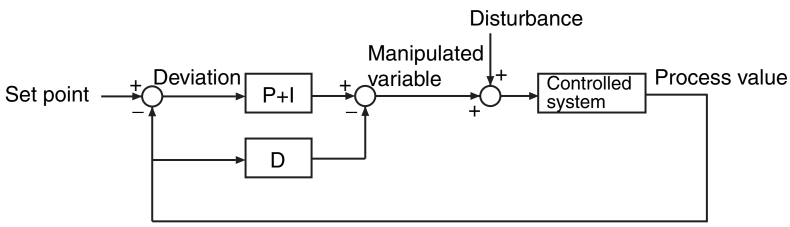


2-PID Control with Set Point Filter

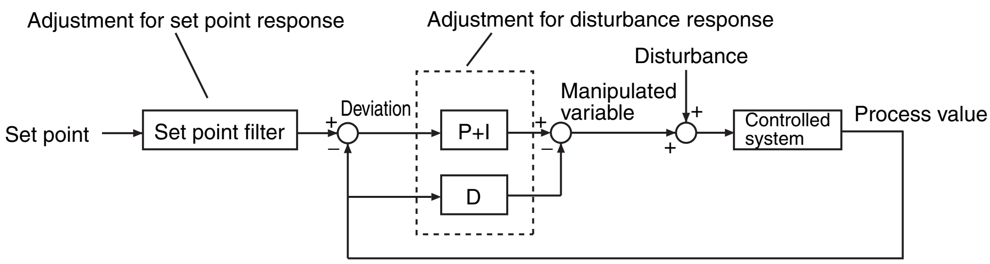
There are three main parameters that you must adjust to perform PID control: the proportional band, integration time, and derivative time. These are called the PID constants. The values of the PID constants affect the following two performances of PID control.

- Set point response: The ability to follow changes in the set point.
- Disturbance response: The ability of correcting the process value for large changes that are caused by disturbances

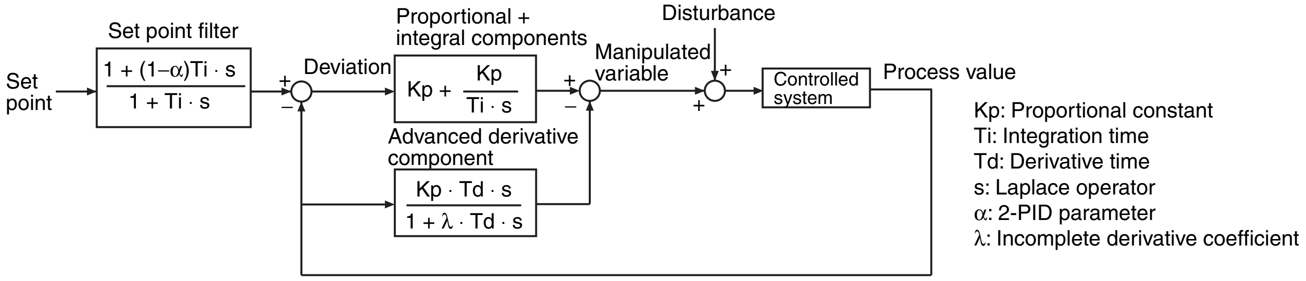
A block diagram for basic PID control is shown below. The set point and disturbance enter the block diagram at different positions. Therefore, finding the optimum PID constants for both set point response performance and disturbance response performance is difficult. In other words, if the PID constants are set for set point response, response to disturbances is slow. If the PID constants are set for disturbance response, overshooting occurs.



To enable both set point response and disturbance response, 2-PID control is used. The 2 in “2-PID” indicates that there are separate parameters to adjust the set point response and the disturbance response. A block diagram for this is shown below. A set point filter that includes an adjustment parameter is added. The PID constants are adjusted to maximize disturbance response. A set point filter adjusts the set point to optimize the set value response for those values. You can adjust the values of the PID constants and the set value of the set point filter independently to increase both the set point response and the disturbance response.



The formulas of the blocks of this instruction are shown below. The set point filter value (i.e., a coefficient for the set point) is adjusted by using the integration time and the 2-PID parameter α. The optimum value of α is 0.65. It normally does not need to be changed. The lower the value of α is, the smaller the influence of the set point filter is.



Starting PID Control

You must use suitable PID constants to execute this instruction. There are the following two ways to achieve this.

● When Suitable PID Constants Are Not Known

Perform autotuning at the start of operation to find suitable PID constants. Change the value of *Run* to TRUE while the value of *StartAT* is TRUE. First, autotuning is executed, and then PID control is started with the PID constants that are found.

● When Suitable PID Constants Are Known

Set suitable PID constants in *ProportionalBand*, *IntegrationTime*, and *DerivativeTime*, and then change *Run* to TRUE. *ProportionalBand*, *IntegrationTime*, and *DerivativeTime* are in-out variables. You cannot set constants for the input parameters. Always define suitable variables, and then assign the values to input parameters.

You can change the PID constants during operation. You can also perform autotuning during operation. To start autotuning during operation, change the value of *StartAT* to TRUE.

Control Status and Manipulated Variable

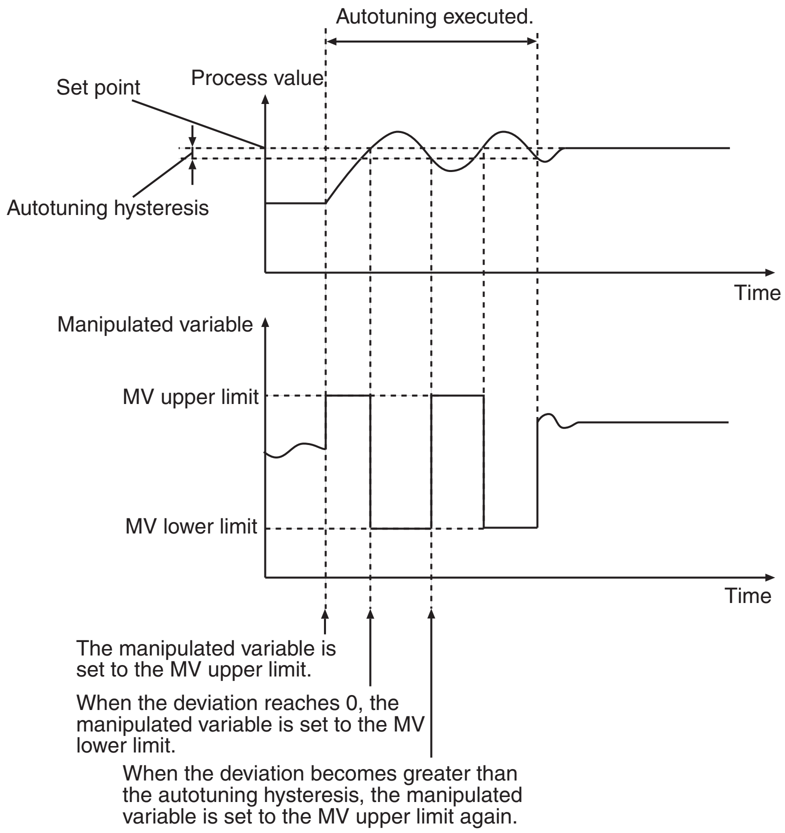
Manipulated variable *MV* is determined according to the control status as shown in the following table.

<table>
<tr><th rowspan="2">Control status</th><th colspan="5">Value of variable</th><th rowspan="2">Manipulated variable MV</th></tr>
<tr><th>ManCtl (manual/auto control)</th><th>Run (execution condition)</th><th>Error (error end)</th><th>MVTrackSw (MV tracking switch)</th><th>ATBusy (autotuning busy)</th></tr>
<tr><td>Error end</td><td rowspan="5">FALSE</td><td rowspan="4">TRUE</td><td>TRUE</td><td>---</td><td rowspan="2">FALSE</td><td>ErrorMV (error MV)</td></tr>
<tr><td>MV tracking during automatic operation</td><td rowspan="3">FALSE</td><td>TRUE</td><td>MVTrackVal (MV tracking value)</td></tr>
<tr><td>Autotuning during automatic operation</td><td rowspan="2">FALSE</td><td>TRUE</td><td>Value repeatedly changes between upper limit of MV and lower limit of MV.</td></tr>
<tr><td>Not autotuning during automatic operation</td><td rowspan="3">FALSE</td><td>Value calculated with current PID constants.</td></tr>
<tr><td>Instruction execution stopped</td><td>FALSE</td><td rowspan="2">---</td><td rowspan="2">---</td><td>StopMV (Stop MV)</td></tr>
<tr><td>Manual operation</td><td>TRUE</td><td>---</td><td>ManMV (manual manipulated variable)</td></tr>
</table>

Autotuning

The 2-PID parameter α is not adjusted very often, so the main parameters that are adjusted for this instruction are the PID constants. The PIDAT instruction supports autotuning of the PID constants. The limit cycle method is used for autotuning. With the limit cycle method, the manipulated variable is temporarily changed to the upper and lower limits of the manipulated variable to find the optimum PID constants based on the resulting changes in the process value. If autotuning is executed when the set point is greater than the process value, the manipulated variable is first set to the upper limit. When the deviation reaches 0, the manipulated variable is set to the lower limit. When the deviation becomes greater than the autotuning hysteresis, the manipulated variable is set to the upper limit again. This process is repeated twice to calculate the optimum PID constants.

If autotuning is executed when the set point is less than the process value, the manipulated variable is first set to the lower limit. Then, the optimum values for the PID constants are calculated with the procedure that is given above.



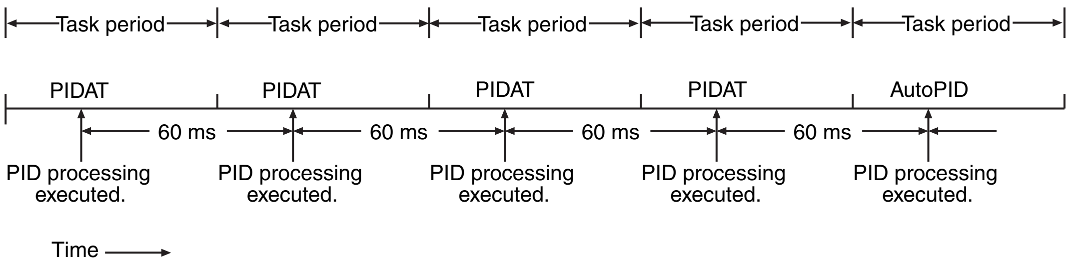
Autotuning is executed during PID control (i.e., when the value of *Run* is TRUE) if the value of *StartAT* changes to TRUE. If *StartAT* is TRUE when *Run* changes to TRUE, autotuning is executed at the start of PID control. When autotuning is completed normally, the calculated PID constants are used immediately. Autotuning is canceled if the value of *StartAT* changes to FALSE during autotuning (i.e., when *ATBusy* is TRUE). If autotuning is canceled, PID control is started again with the previous PID constants.

Execution Timing of PID Control

PID control is repeated periodically. PID processing is performed when the PIDAT instruction is executed in the user program. However, if sampling period *SampTime* has not elapsed since the last time PID processing was performed, PID processing is nor performed. If the elapsed time since the last time PID processing was executed exceeds *SampTime*, the excess time (elapsed time – *SampTime*) is carried forward to the next period. This is shown in the following diagram.

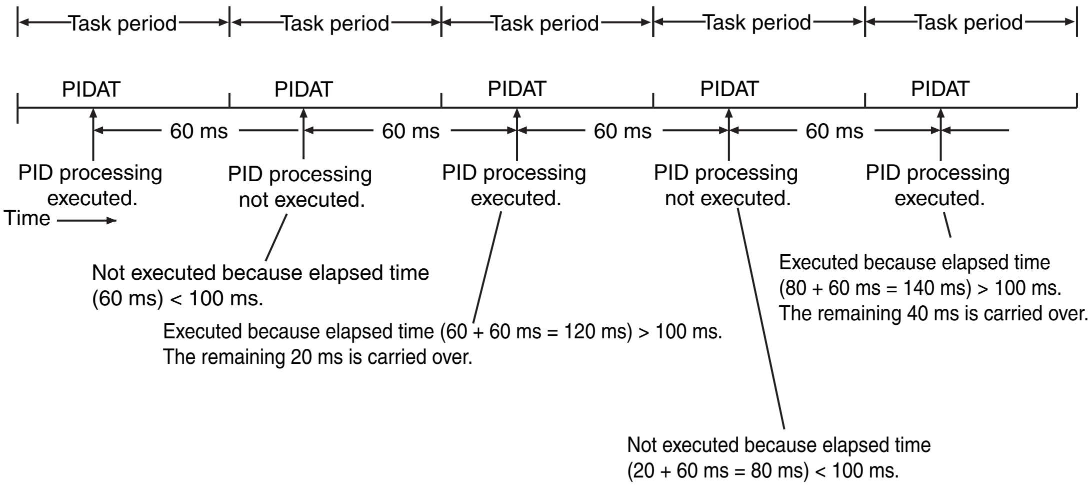
Task period = 60 ms and *SampTime* < 60 ms

The task period is greater than or equal to *SampTime*, so PID processing is executed once every task period.



Task period = 60 ms and *SampTime* = 100 ms

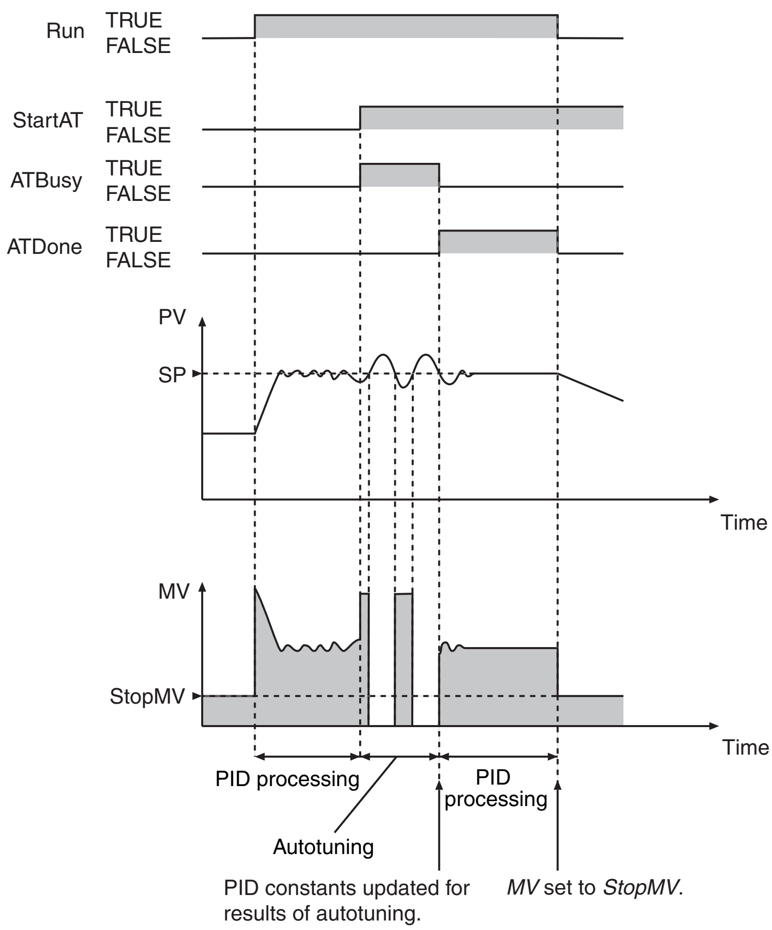
The task period is less than *SampTime*, so DIP processing is not executed every period.



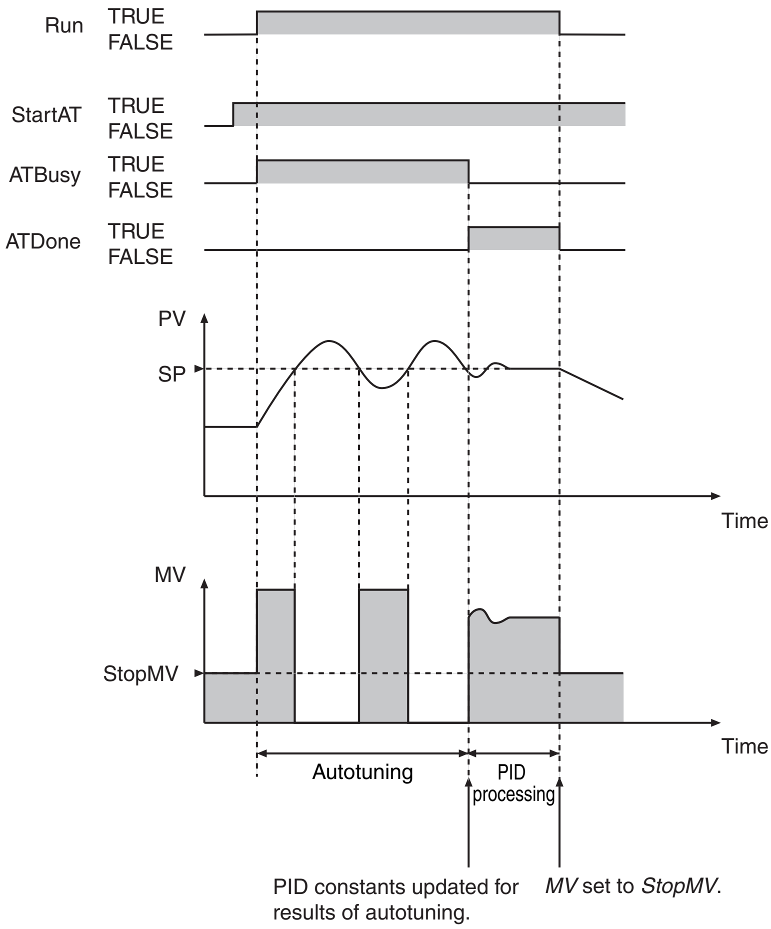
Timing Charts

Timing charts for the instruction variables are provided below for different situations.

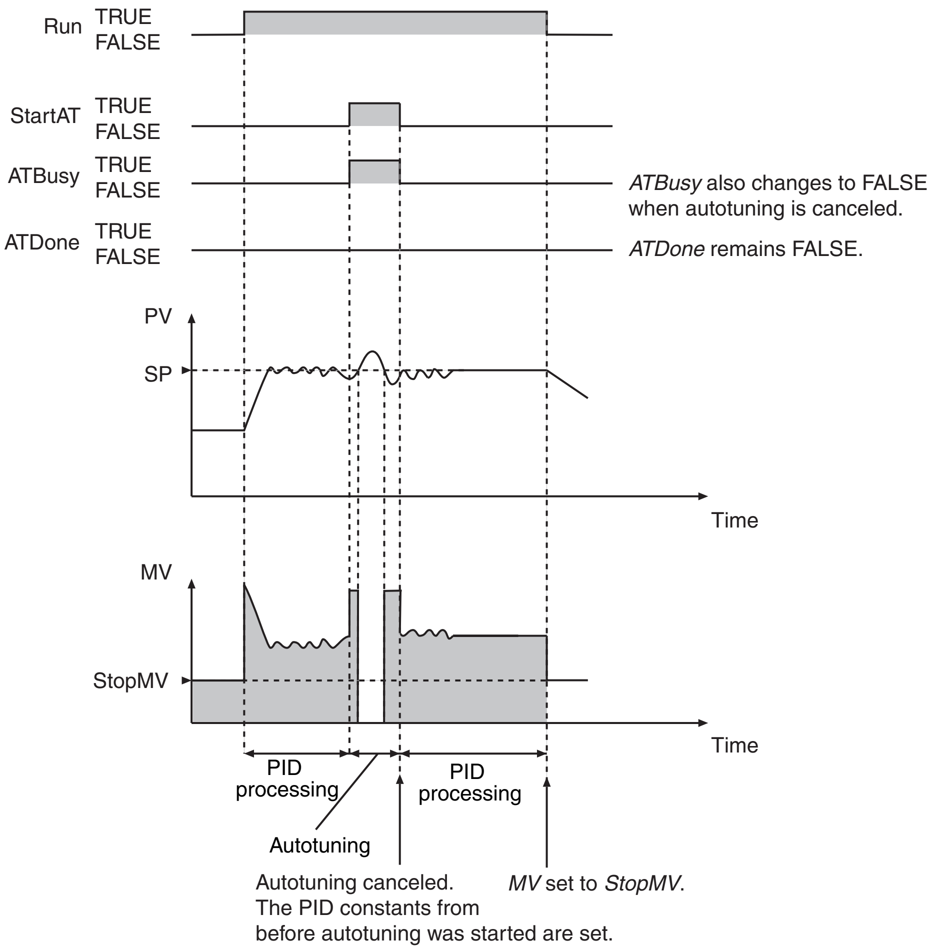
● Autotuning Executed during Automatic Operation



● Autotuning Executed at the Start of PIDAT Execution



- **Autotuning Canceled**

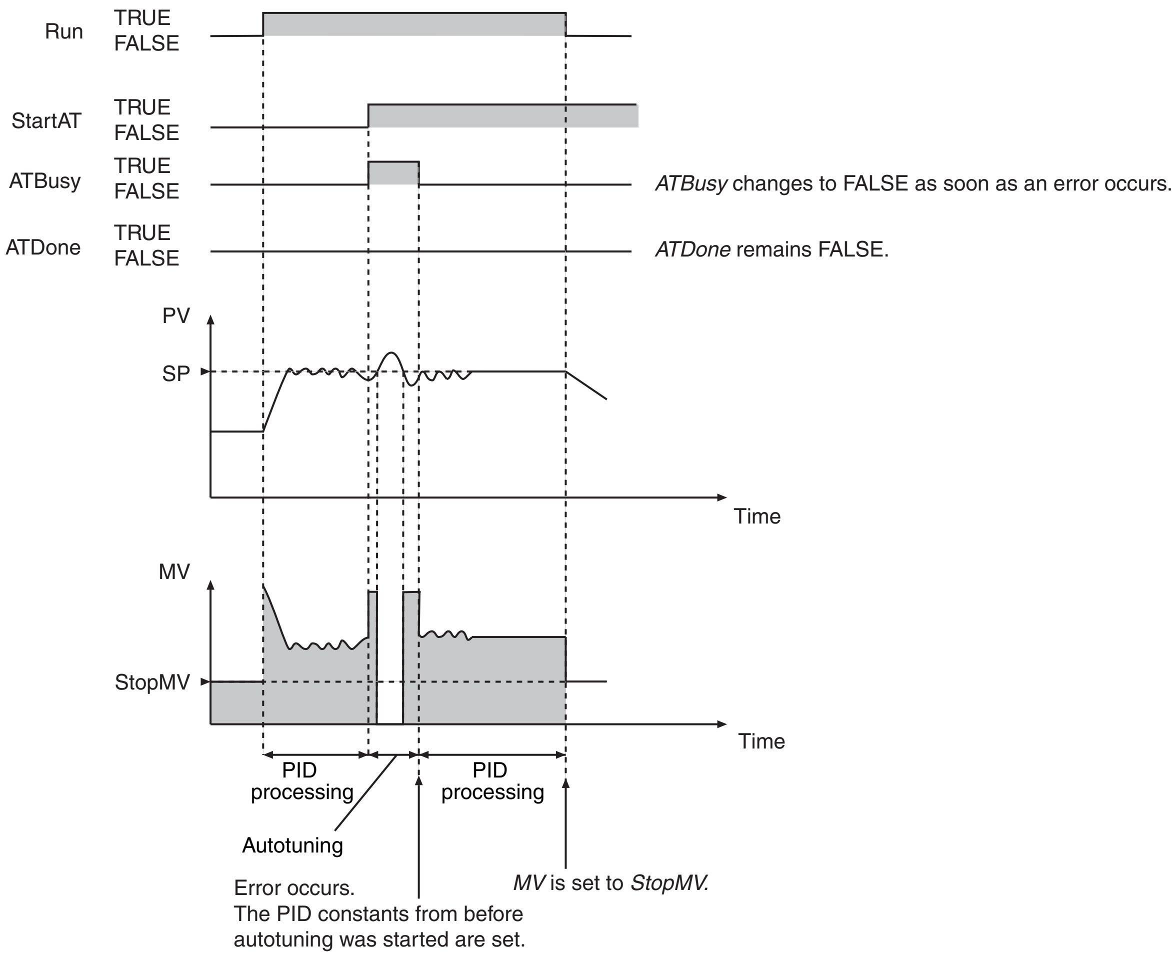


● An Autotuning Error Occurs during Autotuning

An autotuning error occurs and autotuning is stopped in the following cases.

- If the MV equals the MV upper limit and the time for the deviation to reach 0 exceeds 19,999 s.
- If the MV equals the MV lower limit and the time for the deviation to reach *ATHystrs* or higher exceeds 19,999 s.

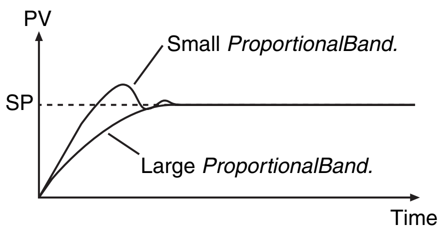
If autotuning is canceled, PID control is started again with the previous PID constants.



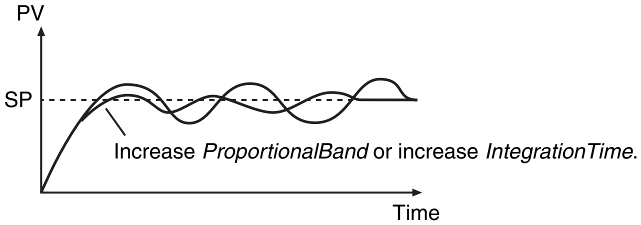
Additional Information

Adjusting PID Constants

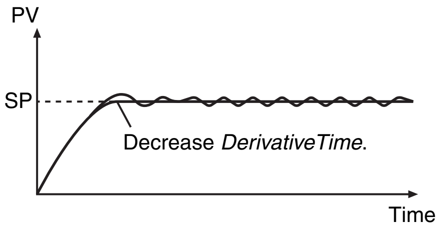
- If you need to eliminate hunting even if it takes time to stabilize the control system, increase the value of *ProportionalBand*. If a certain amount of hunting is not a problem, but it is necessary for the controlled system to stabilize quickly, decrease the value of *ProportionalBand*.



- If hunting continues too long, increase *ProportionalBand* or increase *IntegrationTime*.



- If rapid hunting occurs, decrease *DerivativeTime*.



Initial PID Constants for Temperature Control

If you use the PIDAT instruction for temperature control, use the following initial values of the PID constants as reference. Use the default values for the other variables.

Variables	Initial values (reference values)*
ProportionalBand	10%FS
IntegrationTime	233 s
DerivativeTime	40 s

* If you perform autotuning, use the results from autotuning.

Precautions for Correct Use

- The values of PV and SP must be between the values of *RngLowLmt* and *RngUpLmt*, inclusive. Align the units of these variables as shown below.

Unit	Values of *PV* and *SP*	Values of *RngLowLmt* and *RngUpLmt*
% FS	PV = (Process value in physical units – MIN)/(MAX – MIN) × 100 SP = (Set point in physical units – MIN)/(MAX – MIN) × 100*	RngLowLmt = 0 RngUpLmt = 100
Physical unit	PV = Process value in physical units SV = Set point in physical units	RngLowLmt = MIN RngUpLmt = MAX*

* MAX: Upper limit of input range in physical units, MIN: Lower limit of input range in physical units,

- The following table shows which variables can be changed depending on the operating status.

Variables	Control status		
	Instruction execution stopped*1	Automatic operation when autotuning is not being executed*2	Automatic operation when autotuning is being executed*3
Run	Possible	Possible	Possible
ManCtl	Possible	Possible	Possible
StartAT	Possible	Possible	Possible
PV	Possible	Possible	Possible
SP	Possible	Possible	Not possible
MVLowLmt	Possible	Possible	Not possible
MVUpLmt	Possible	Possible	Not possible
ManResetVal	Possible	Possible	Not possible

Variables	Control status		
	Instruction execution stopped*1	Automatic operation when autotuning is not being executed*2	Automatic operation when autotuning is being executed*3
MVTrackSw	Possible	Possible	Not possible
MVTrackVal	Possible	Possible	Not possible
StopMV	Possible	Possible	Possible
ErrorMV	Possible	Possible	Possible
Alpha	Possible	Possible	Not possible
ATCalcGain	Possible	Possible	Not possible
ATHystrs	Possible	Possible	Not possible
SampTime	Possible	Not possible	Not possible
RngLowLmt	Possible	Not possible	Not possible
RngUpLmt	Possible	Not possible	Not possible
DirOpr	Possible	Not possible	Not possible
ProportionalBand	Possible	Possible	Not possible
IntegrationTime	Possible	Possible	Not possible
DerivativeTime	Possible	Possible	Not possible
ManMV	Possible	Possible	Possible

*1 *ManCtl* is TRUE, *Run* is FALSE, *Error* is TRUE, or *MVTrackSw* is TRUE.

*2 *ManCtl* is FALSE, Run is TRUE, Error is FALSE, MVTrackSw is FALSE, and ATBusy is FALSE.

*3 *ManCtl* is FALSE, Run is TRUE, Error is FALSE, MVTrackSw is FALSE, and ATBusy is TRUE.

- *SampTime* is truncated below 100 nanoseconds.
- If the value of *StartAT* changes to TRUE while the value of *ManCtl* is TRUE, autotuning starts the next time the value of *ManCtl* changes to FALSE.
- If the value of *ErrorMV* is not within the valid range (–320 to 320), the value of *MV* will be 0 when an error occurs.
- Autotuning is canceled if the value of *ManCtl* changes to TRUE during autotuning.
- The value of *Error* does not change to TRUE even if an error occurs during autotuning.
- An error occurs in the following case. *Error* will change to TRUE, and an error code is assigned to *ErrorID*. *ATDone* and *ATBusy* change to FALSE. *MV* is set to the value of *ErrorMV* if the values of *ManCtl* and *Run* are FALSE. If the value of *ErrorMV* is outside of the valid range, the value of *MV* is 0.

Error	Value of *ErrorID*
The value of an input variable is outside of the valid range.	16#0400
RngLowLmt is greater than or equal to *RngUpLmt*.	16#0401
MVLowLmt is greater than or equal to *MVUpLmt*.	

- If an error stop is required for conditions other than the above, program the system so that the value of *Run* changes to FALSE when the error occurs.
- If an error occurs because the value of *PV* or *SP* exceeds the valid range, the error status is maintained for five seconds even if the value returns to within the valid range sooner. That is, the value of *Error* will remain FALSE for five seconds.
- PID control is restarted automatically if the value of *Run* is TRUE after the error is reset. Autotuning is restarted automatically if the values of *Run* and *StartAT* are TRUE.
- A check is made for errors each sampling period.

Sample Programming

In this sample, the PIDAT instruction is used to perform temperature control. The manipulated variable of the PIDAT instruction is converted to a time-proportional value and output to a heating device. This sample uses a timer instruction to convert to a time-proportional value. To use the TimeProportionalOut instruction to convert to a time-proportional value, refer to the sample programming that is provided for the TimeProportionalOut instruction (page 2-733).

Specifications

Temperature control is performed according to the following specifications.

Item	Specification
Input type	K thermocouple
Input Unit	CJ1W-PH41U Isolated-type Universal Input Unit
Output Unit	CJ1W-OD212 Transistor Output Unit
Set point	90°C
Sampling period for PID control	100 ms
Output control period	1 s

Configuration and Settings

The following setting is used for the CJ1W-PH41U Input Unit.

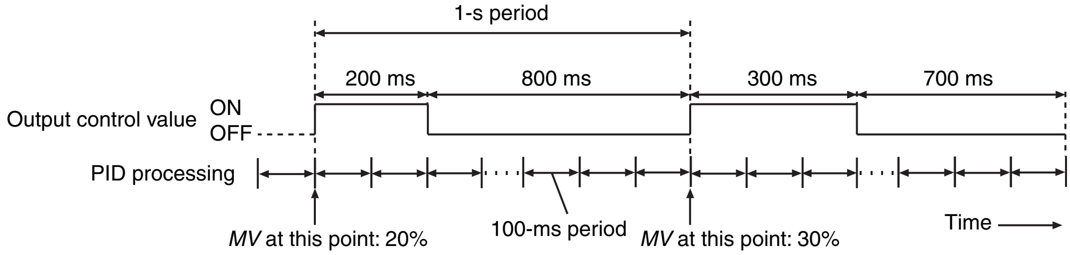
Setting	Set value
Input1:Input signal type	K(1)

The following I/O map settings are used.

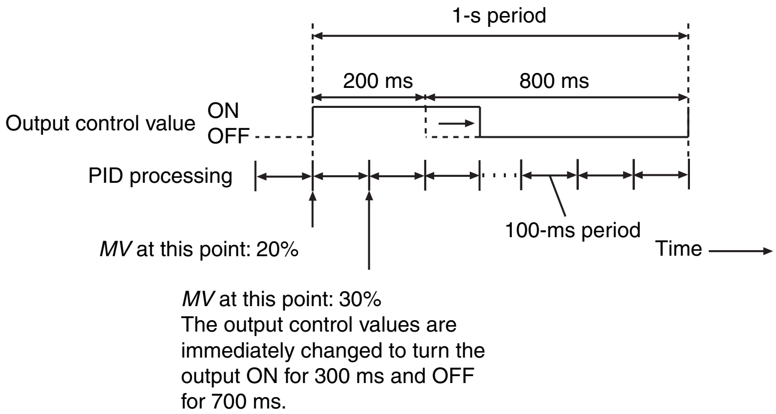
Unit	I/O port	Description	Variable
CJ1W-PH41U	Ch1_AIInPV	Process value for input 1 (INT data)	AI1
CJ1W-OD212	Ch1_Out00	Bit 00 of output word 1	DO1

Processing

- The manipulation value MV of the PIDAT instruction is obtained to control the output to the temperature controller. The output to the temperature controller is turned ON and OFF.
- The sampling period (*InitSetParams.SampTime*) of the PIDAT instruction is set to 100 ms. The task period must be sufficiently shorter than 100 ms. Therefore, the value of MV is refreshed every 100 ms.
- The output control period is 1 s. During that period, the ON time and OFF time of the output control value are controlled with a time-proportional output. For example, if the obtained value of *MV* is 20%, the output to the temperature control is ON for 200 ms and OFF for 800 ms. This is repeated at a 1-s period.



- If the most recent value of *MV* is smaller than the value of *MV* when the output control values were determined, the output control values do not change. If the most recent value of *MV* is larger than the value of *MV* when the output control values were determined, the most recent value is immediately reflected in the output control values. For example, assume that the output control values were determined when the value of *MV* was 20% (ON 200 ms, OFF 800 ms). If after 100 ms, the new value of *MV* is 30%, the output control values are immediately changed to turn the output ON for 300 ms and OFF for 700 ms.



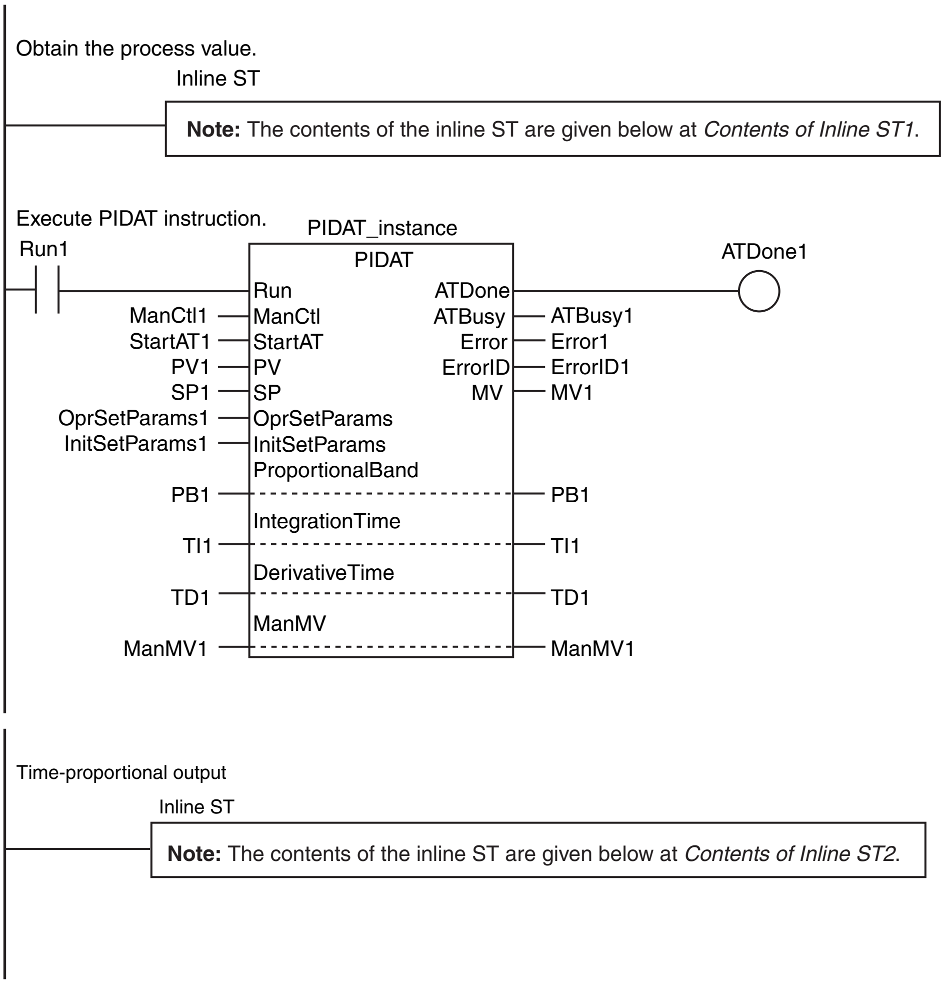
- If autotuning is performed and the value of MV is 100%, the output is immediately turned ON regardless of the control period.

Application Programming

LD

Variable	Data type	Initial value	Retain	Comment
Run1	BOOL	FALSE		Execution condition
ManCtl1	BOOL	FALSE		Manual/auto control
StartAT1	BOOL	FALSE		Autotuning execution condition
PV1	REAL	0.0		Process value
SP1	REAL	90		Set point
OprSetParams1	_sOPR_SET_PARAMS	(MVLowLmt:=0.0, MVUpLmt:=100.0, ManResetVal:=0.0, MVTrackSw:=FALSE, MVTrackVal:=0.0, StopMV:=0.0, ErrorMV:=0.0, Alpha:=0.65, ATCalcGain:=1.0, ATHystrs:=0.2)		Operation setting parameters
InitSetParams1	_sINIT_SET_PARAMS	(SampTime:=T#100ms, RngLowLmt:=0.0, RngUpLmt:=1000.0, DirOpr:=FALSE)		Initial setting parameters
PB1	REAL	10	✔	Proportional band
TI1	TIME	T#0S	✔	Integration time
TD1	TIME	T#0S	✔	Derivative time
ManMV1	REAL	0.0		Manual manipulated variable
ATDone1	BOOL	FALSE		Autotuning normal completion
ATBusy1	BOOL	FALSE		Executing autotuning
Error1	BOOL	FALSE		Error

Variable	Data type	Initial value	Retain	Comment
ErrorID1	WORD	16#0		Error ID
MV1	REAL	0.0		Manipulated variable
PulseOnTime	TIME	T#0s		Control output ON time
PulseCycTime	TIME	T#1s		Control period
ResetPulse	BOOL	FALSE		Timer reset
PIDAT_instance	PIDAT			
TOF_instance	TOF			
TON_instance	TON			



Contents of Inline ST1

```
PV1:=INT_TO_REAL(AI1)/REAL#10.0;        // Convert PV AI1 to real number.
                                        // CJ1W-PH41U output is ten times the process value, so divide by 10.0.
```

Contents of Inline ST2

```
PulseOnTime:=MULTIME(PulseCycTime, MV1/REAL#100.0);           // Calculate ON time output control value.
TOF_instance(In:=BOOL#FALSE, PT:=PulseOnTime, Q=>DO1);        // Switch between ON and OFF with TOF  instruction.
TON_instance(In:=BOOL#TRUE, PT:=PulseCycTime, Q=>ResetPulse); // Measure timer reset time with TON instruction.
IF (ResetPulse=BOOL#TRUE) THEN                                // Reset timer.
       TOF_instance(In:=BOOL#TRUE);
       TON_instance(In:=BOOL#FALSE);
END_IF;
IF ( (ATBusy1=BOOL#TRUE) & (MV1=REAL#100.0) ) THEN            // If MV1 = 100% for autotuning...
       DO1:=BOOL#TRUE;                                        // Turn ON the output immediately.
END_IF;
```

ST

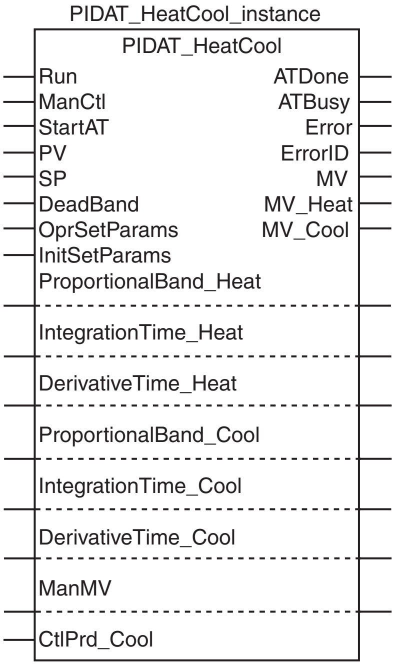
Variable	Data type	Initial value	Retain	Comment
Run1	BOOL	FALSE		Execution condition
ManCtl1	BOOL	FALSE		Manual/auto control
StartAT1	BOOL	FALSE		Autotuning execution condition
PV1	REAL	0.0		Process value
SP1	REAL	90		Set point
OprSetParams1	_sOPR_SET_PARAMS	(MVLowLmt:=0.0, MVUpLmt:=100.0, ManResetVal:=0.0, MVTrackSw:=FALSE, MVTrackVal:=0.0, StopMV:=0.0, ErrorMV:=0.0, Alpha:=0.65, ATCalcGain:=1.0, ATHystrs:=0.2)		Operation setting parameters
InitSetParams1	_sINIT_SET_PARAMS	(SampTime:=T#100ms, RngLowLmt:=0.0, RngUpLmt:=1000.0, DirOpr:=FALSE)		Initial setting parameters
PB1	REAL	10	✔	Proportional band
TI1	TIME	T#0S	✔	Integration time
TD1	TIME	T#0S	✔	Derivative time
ManMV1	REAL	0.0		Manual manipulated variable
ATDone1	BOOL	FALSE		Autotuning normal completion
ATBusy1	BOOL	FALSE		Executing autotuning
Error1	BOOL	FALSE		Error
ErrorID1	WORD	16#0		Error ID
MV1	REAL	0.0		Manipulated variable
PulseOnTime	TIME	T#0s		Control output ON time
PulseCycTime	TIME	T#1s		Control period
ResetPulse	BOOL	FALSE		Timer reset
PIDAT_instance	PIDAT			
TOF_instance	TOF			
TON_instance	TON			

```
// Convert PV AI1 to real number.
PV1:=INT_TO_REAL(AI1)/REAL#10.0;   // CJ1W-PH41U output is ten times the process value, so divide by 10.0.

// Execute PIDAT instruction.
PIDAT_instance(
    Run              :=Run1,
    ManCtl           :=ManCtl1,
     StartAT          :=StartAT1,
     PV               :=PV1,
     SP               :=SP1,
     OprSetParams     :=OprSetParams1,
     InitSetParams    :=InitSetParams1,
     ProportionalBand :=PB1,
     IntegrationTime  :=TI1,
     DerivativeTime   :=TD1,
     ManMV            :=ManMV1,
     ATDone            =>ATDone1,
     ATBusy            =>ATBusy1,
```

```
     Error              =>Error1,
     ErrorID            =>ErrorID1,
    MV                 =>MV1);

// Time-proportional output
PulseOnTime:=MULTIME(PulseCycTime, MV1/REAL#100.0);
// Calculate ON time output control value.
TOF_instance(In:=BOOL#FALSE, PT:=PulseOnTime, Q=>DO1);
// Switch between ON and OFF with TOF instruction.
TON_instance(In:=BOOL#TRUE, PT:=PulseCycTime, Q=>ResetPulse);
// Measure timer reset time with TON instruction.
IF (ResetPulse=BOOL#TRUE) THEN                        // Reset timer.
    TOF_instance(In:=BOOL#TRUE);
    TON_instance(In:=BOOL#FALSE);
END_IF;
IF ( (ATBusy1=BOOL#TRUE) & (MV1=REAL#100.0) ) THEN    // If MV1 = 100% for autotuning...
    DO1:=BOOL#TRUE;                                   // Turn ON the output immediately.
END_IF;
```

PIDAT_HeatCool

The PIDAT_HeatCool instruction performs heating/cooling PID control with autotuning (2-PID control with set point filter).

Instruction	Name	FB/FUN	Graphic expression	ST expression
PIDAT_HeatCool	Heating/Cooling PID with Autotuning	FB		PIDAT_HeatCool_instance(Run, ManCtl, StartAT, PV, SP, DeadBand, OprSetParams, InitSetParams, ProportionalBand_Heat, IntegrationTime_Heat, DerivativeTime_Heat, ProportionalBand_Cool, IntegrationTime_Cool, DerivativeTime_Cool, ManMV, CtlPrd_Cool, ATDone, ATBusy, Error, ErrorID, MV, MV_Heat, MV_Cool);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Run	Execution condition	Input	TRUE: Execute FALSE: Stop	Depends on data type.	---	FALSE
ManCtl	Manual/auto control		TRUE: Manual operation FALSE: Automatic operation			
StartAT	Autotuning execution condition		TRUE: Execute FALSE: Cancel			
PV	Process value		Process value	*1		0
SP	Set point		Set point			
DeadBand	Deadband		Deadband/overlap band setting	–320.0 to 320.0	%	
OprSetParams	Operation setting parameters		Parameters set during operation	---	---	---
InitSetParams	Initial setting parameters		Initial setting parameters			
CtlPrd_Cool	Cooling control period		Control period when time-proportional output is used for *MV_Cool*	T#0.1s to T#100s		T#20s
ProportionalBand_Heat	Proportional band for heating control	In-out	Proportional band for heating control	0.01 to 1000.00	%FS	---
IntegrationTime_Heat	Integration time for heating control		Integration time for heating control The higher the value is, the weaker the integral action is. No integral action is performed for 0.	T#0.0000s to T#10000.0000s*2	s	
DerivativeTime_Heat	Derivative time for heating control		Derivative time for heating control The higher the value is, the stronger the derivative action is. No derivative action is performed for 0.	T#0.0000s to T#10000.0000s*2		
ProportionalBand_Cool	Proportional band for cooling control		Proportional band for cooling control	0.01 to 1000.00	%FS	
IntegrationTime_Cool	Integration time for cooling control		Integration time for cooling control The higher the value is, the weaker the integral action is. No integral action is performed for 0.	T#0.0000s to T#10000.0000s*2	s	
DerivativeTime_Cool	Derivative time for cooling control		Derivative time for cooling control The higher the value is, the stronger the derivative action is. No derivative action is performed for 0.	T#0.0000s to T#10000.0000s*2		
ManMV	Manual manipulated variable		Manual manipulated variable	–320 to 320	%	

Name	Meaning	I/O	Description	Valid range	Unit	Default
ATDone	Autotuning normal completion	Output	TRUE: Normal completion FALSE: *3	Depends on data type.	---	---
ATBusy	Autotuning busy		TRUE: Autotuning FALSE: Not autotuning			
MV	Manipulated variable		Manipulated variable		%	
MV_Heat	Manipulated variable for heating control		Manipulated variable for heating control	0 to 320		
MV_Cool	Manipulated variable for cooling control		Manipulated variable for cooling control	0 to 320		

*1 Value of input range lower limit *InitSetParams.RngLowLmt* to Value of input range upper limit *InitSetParams.RngUpLmt*

*2 Digits below 0.0001 s are truncated.

*3 FALSE indicates an error end, that PID control is in progress without autotuning, or that PID control is not in progress.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Run	OK																			
ManCtl	OK																			
StartAT	OK																			
PV														OK						
SP														OK						
DeadBand														OK						
OprSet Params	Refer to *Function* for details on the structure _sOPR_SET_PARAMS.																			
InitSet Params	Refer to *Function* for details on the structure _sINIT_SET_PARAMS.																			
CtlPrd _Cool																OK				
Proportional Band_Heat														OK						
Integration-Time_Heat																OK				
Derivative Time_Heat																OK				
Proportional Band_Cool														OK						
Integration-Time_Cool																OK				
Derivative Time_Cool																OK				
ManMV														OK						
ATDone	OK																			
ATBusy	OK																			
MV														OK						
MV_Heat														OK						
MV_Cool														OK						

Function

The PIDAT_HeatCool instruction performs heating/cooling PID control of a manipulated variable for a temperature controller or other device.

Heating/cooling PID control is started when the value of execution condition *Run* changes to TRUE. While the value of *Run* is TRUE, the following process is repeated periodically: process value *PV* is read, heating/cooling PID processing is performed, and manipulated variable for heating *MV_Heat* and manipulated variable for cooling *MV_Cool* are output.

Heating/cooling PID control is stopped when the value of *Run* changes to FALSE.

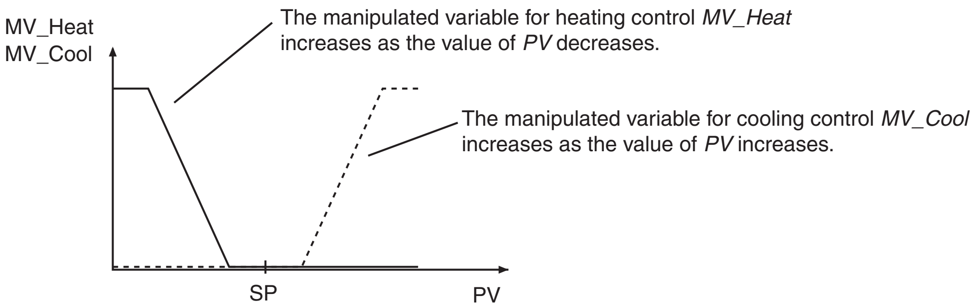
Autotuning is supported to automatically find the optimum PID constants for heating control and for cooling control.

When the value of the autotuning execution condition *StartAT* changes to TRUE, the PID constants for heating control and cooling control are autotuned.

Difference between the PIDAT_HeatCool and PIDAT Instructions

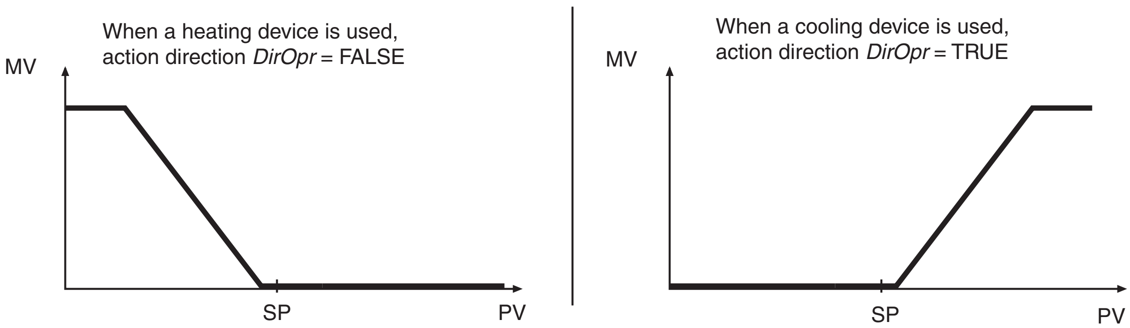
PIDAT_HeatCool Instruction

The PIDAT_HeatCool instruction uses both a heating device and a cooling device to control the temperature. Therefore, manipulated variables are output for two different control operations: the manipulated variable for heating control, *MV_Heat*, and the manipulated variable for cooling control, *MV_Cool*. Autotuning finds the optimum PID constants for heating control and the optimum PID constants for cooling control.



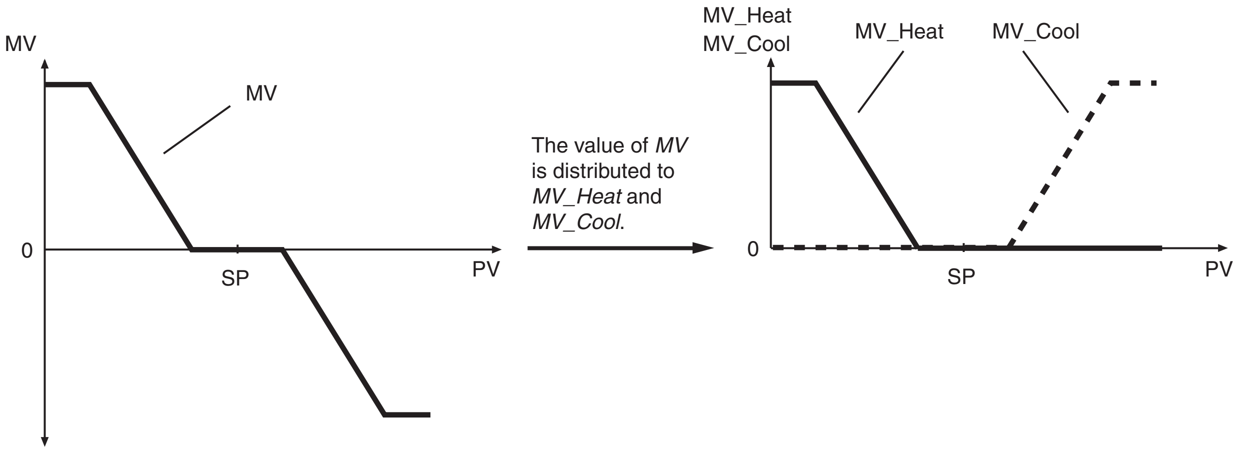
PIDAT Instruction

The PIDAT instruction uses either a heating device or a cooling device to control the temperature. Therefore, only one manipulated variable (*MV*) is output. Also, there is a parameter, action direction *DirOpr*, that determines whether the manipulated variable is output to a heating device or to a cooling device. The PIDAT_HeatCool instruction does not use *DirOpr*.



Manipulated Variable *MV* Compared with Manipulated Variable for Heating Control *MV_Heat* and Manipulated Variable for Cooling Control *MV_Cool*

MV is the manipulated variable for an instruction like the PIDAT instruction that uses either a heating device or a cooling device to control the temperature. The PIDAT_HeatCool instruction also calculates *MV* in the same way as the PIDAT instruction. *MV* is then distributed to the manipulated variable for heating and the manipulated variable for cooling. The following figure shows conceptually how the value of *MV* is distributed to *MV_Heat* and *MV_Cool.* The value of *MV_Cool* is the absolute value of *MV* when *MV* is negative.



The above figure is conceptual. The actual values of *MV_Heat* and *MV_Cool* are not exactly the negative and positive values of *MV.* The values of *MV_Heat* and *MV_Cool* are calculated from special formulas based on the value of *MV.*

Structure Specifications

The data type of operation setting parameter *OprSetParams* is structure _sOPR_SET_PARAMS. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
OprSetParams	Operation Setting Parameters	Parameters that are set during operation.	_sOPR_SET_PARAMS	---	---	---
MVLowLmt	MV Lower Limit	Lower limit of *MV_Heat* and *MV_Cool*	REAL	–320 to 320*1	%	–100
MVUpLmt	MV Upper Limit	Upper limit of *MV_Heat* and *MV_Cool*	REAL	–320 to 320*1	%	100
ManResetVal	Manual Reset Value	Not used.	REAL	–320 to 320	%	0
MVTrackSw	MV Tracking Switch	MV Tracking Switch TRUE: ON FALSE: OFF	BOOL	Depends on data type.	---	FALSE
MVTrackVal	MV Tracking Value	The value that is set in *MV* during MV tracking.	REAL	–320 to 320	%	0
StopMV	Stop MV	The value that is set in *MV* when instruction execution is stopped.	REAL	–320 to 320	%	0
ErrorMV	Error MV	The value that is set in *MV* when an error occurs.	REAL	–320 to 320	%	0
Alpha	2-PID Parameter α	The set point filter is disabled if the set point filter coefficient α is 0.	REAL	0.00 to 1.00	---	0.65
ATCalcGain	Autotuning Calculation Gain	Adjustment coefficient from autotuning results. Stability is given higher priority with higher values. The speed of response is given higher priority with lower values.	REAL	0.1 to 10.0	---	0.8
ATHystrs	Autotuning Hysteresis	The hysteresis of the limit cycle.	REAL	0.01 to 10.0	%FS	0.05

*1 *MVLowLmt* must be less than *MVUpLmt*.

The data type of initial setting parameter *InitSetParams* is structure _sINIT_SET_PARAMS. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
InitSetParams	Initial Setting Parameters	Initial setting parameters.	_sINIT_SET_PARAMS	---	---	---
SampTime	Sampling Period	The period for PID processing.	TIME	T#0.0001s to #100.0000s	s	T#0.05s
RngLowLmt	Lower Limit of Input Range	The lower limit of *PV* and *SP*.	REAL	–32000 to 32000*1	---	0
RngUpLmt	Upper Limit of Input Range	The upper limit of *PV* and *SP*.	REAL			100
DirOpr	Action Direction	Not used.	BOOL	Depends on data type.		FALSE

*1 *RngLowLmt* must be less than *RngUpLmt*.

Meanings of Variables

The meanings of the variables that are used in this command are described below.

● *Run* (Execution Condition)

This is the execution condition for the instruction.

Heating/cooling PID control is performed while the value is TRUE. Heating/cooling PID control is stopped when the value changes to FALSE.

● *ManCtl* (Manual/Auto Control)

This instruction can be executed in one of two modes: Manual operation or automatic operation. The value of *ManCtl* determines which mode is used.

Value of *ManCtl*	Operation mode	Value of *MV*
TRUE	Manual	Value of *ManMV* (Heating/cooling PID control is not performed.)
FALSE	Automatic	Value that is calculated for heating/cooling PID control

● *StartAT* (Autotuning Execution Condition)

This is the execution condition for autotuning the PID constants.
If the value of *StartAT* is TRUE when the value of *Run* changes to TRUE, autotuning is performed when PID control is started.
If the value of *StartAT* changes to TRUE during heating/cooling PID control (i.e., when the value of *Run* is TRUE), autotuning is performed during heating/cooling PID control.
In either case, autotuning is canceled if the value of *StartAT* changes to FALSE during autotuning.
Autotuning is described in more detail later.

● *PV* (Process Value)

This is the process value of the controlled system.

● *SP* (Set Point)

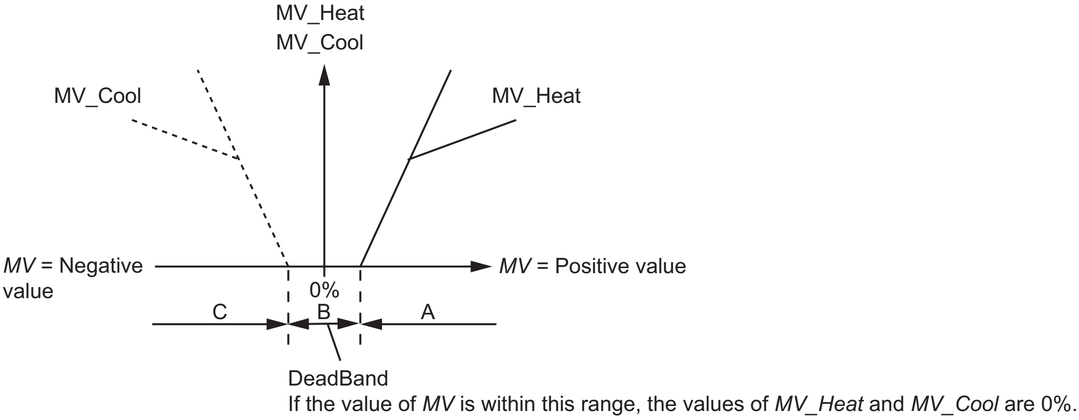
This is the set point for the controlled system.

● *DeadBand* (Deadband)

DeadBand determines how the value of *MV* is distributed to *MV_Heat* and *MV_Cool*. *DeadBand* gives the range of the value of *MV* centered on an *MV* value of 0 within which both heating and cooling control operations are not performed.

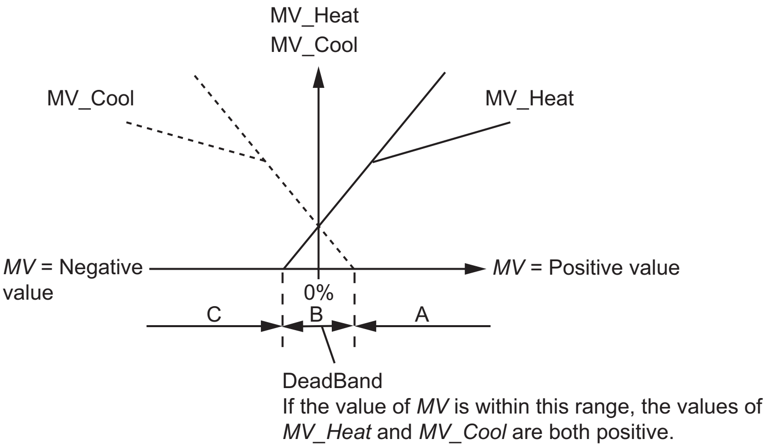
The following table and figure show the relationship between the value of *MV* and the values of *MV_Heat* and *MV_Cool*.

Value of *MV*	Value of *MV_Heat*	Value of *MV_Cool*
Larger than the deadband (Area A)	Positive. Increases as the value of *MV* increases.	0
Within the deadband (Area B)	0	0
Smaller than the deadband (Area C)	0	Positive. Increases as the value of *MV* decreases.



You can also set a negative value for *DeadBand*. If the value of *DeadBand* is negative while the value of *MV* is within the deadband, both heating and cooling control are performed. The following table and figure show the relationship between the value of *MV* and the values of *MV_Heat* and *MV_Cool* when the value of *DeadBand* is negative.

Value of *MV*	Value of *MV_Heat*	Value of *MV_Cool*
Larger than the deadband (Area A)	Positive. Increases as the value of *MV* increases.	0
Within the deadband (Area B)	Positive. Increases as the value of *MV* increases.	Positive. Increases as the value of *MV* decreases.
Smaller than the deadband (Area C)	0	Positive. Increases as the value of *MV* decreases.



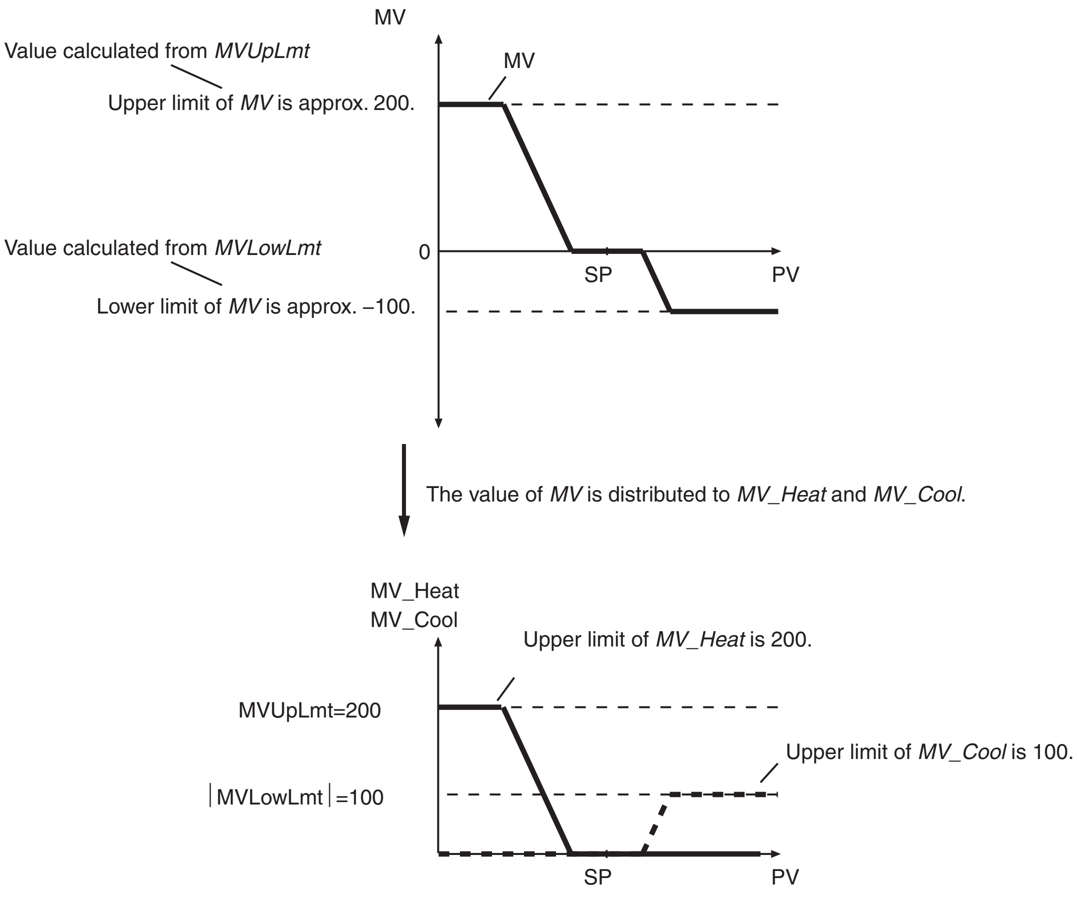
- ***MVLowLmt* (MV Lower Limit) and *MVUpLmt* (MV Upper Limit)**

You can limit the values of *MV_Heat* and *MV_Cool*. The upper and lower limits of *MV_Heat* and *MV_Cool* are determined by *MVLowLmt* and *MVUpLmt*. The following procedure is used to find the values of *MV_Heat* and *MV_Cool*.

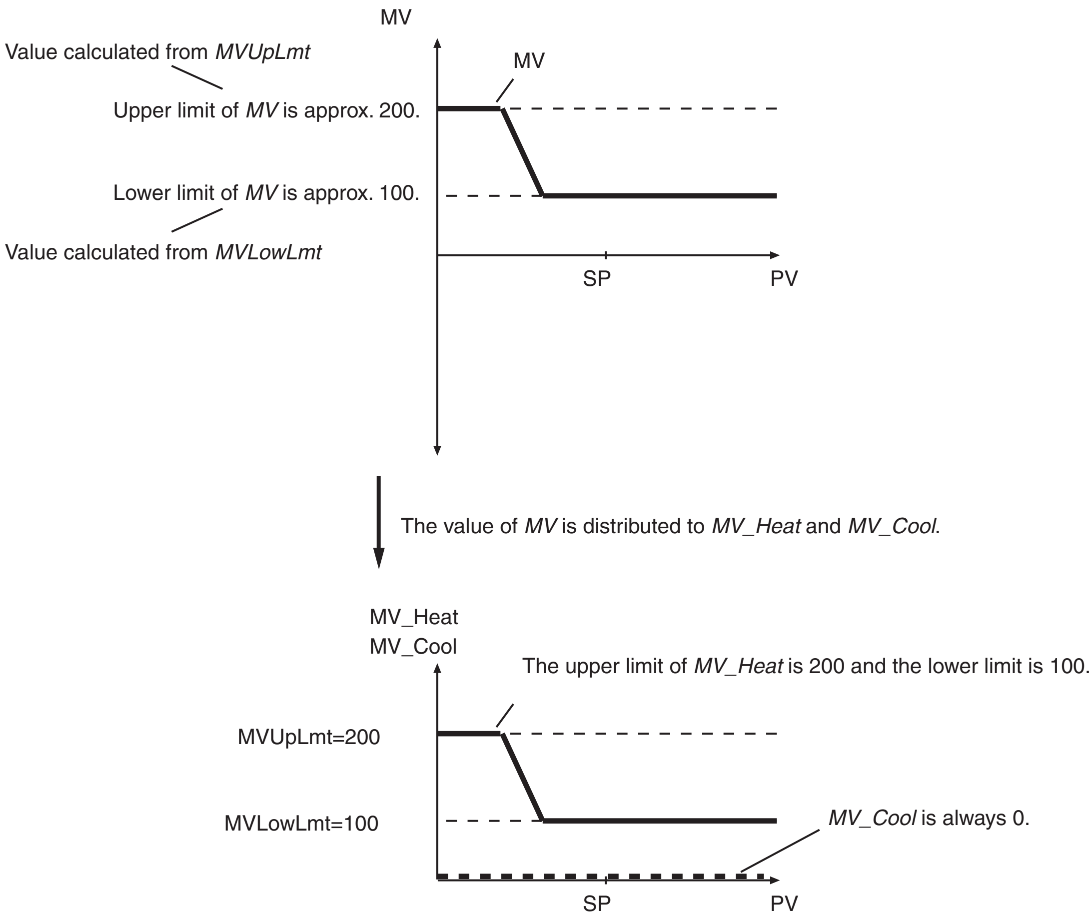
1 The heating/cooling PID processing is performed to find *MV.* The upper and lower limits of *MV* are calculated from special formulas based on *MVLowLmt* and *MVUpLmt.*

2 *MV_Heat* and *MV_Cool* are found by distributing *MV*.

The following figure shows the relationship between *MV*, *MV_Heat,* and *MV_Cool* when *MVLowLmt* is –100 and *MVUpLmt* is 200. The calculated upper limit of *MV_Heat* is 200 and the calculated lower limit is 0. The calculated upper limit of *MV_Cool* is 100 and the calculated lower limit is 0. In other words, the upper limit of *MV_Heat* is the same as the value of *MVUpLmt,* but the upper limit of *MV_Cool* is the absolute value of *MVLowLmt.*



The following figure shows the relationship between *MV*, *MV_Heat*, and *MV_Cool* when *MVLowLmt* is 100 and *MVUpLmt* is 200. The calculated upper limit of *MV_Heat* is 200 and the calculated lower limit is 100. The value of *MV_Cool* is always 0. In other words, the upper and lower limits of *MV_Heat* are the same as *MVUpLmt* and *MVLowLmt*.



As shown above, the upper and lower limits of *MV_Heat* and *MV_Cool* change as shown in the following table depending on whether *MVLowLmt* and *MVUpLmt* are positive values or negative values.

Value of *MVLowLmt*	Value of *MVUpLmt*	*MV_Heat*		*MV_Cool*	
		Lower limit	Upper limit	Lower limit	Upper limit
Positive	Positive	MVLowLmt	MVUpLmt	0	0
Negative	Positive	0	MVUpLmt	0	Absolute value of *MVLowLmt*
Negative	Negative	0	0	Absolute value of *MVUpLmt*	Absolute value of *MVLowLmt*

Always set *MVLowLmt* and *MVUpLmt* so that *MVLowLmt* is less than *MVUpLmt*. Also, if *MV* is set to *StopMV*, *ErrorMV*, or *ManMV*, limit control is not applied.

You can change *MVLowLmt* and *MVUpLmt* even if the control status of this instruction is not auto-tuning during automatic operation.

However, if you change *MVLowLmt* or *MVUpLmt* to an expansion direction during operation, the value of *MV_Heat* or *MV_Cool* which is the same as one in the last sampling period is output and changed smoothly at this time (bumpless).

Repeated changing of *MVLowLmt* or *MVUpLmt* will affect the control performance so that sufficient control performance may not obtain.

Confirm the effects on the control performance before you repeatedly change *MVLowLmt* or *MVUpLmt* during operation.

● *ManResetVal* (Manual Reset Value)

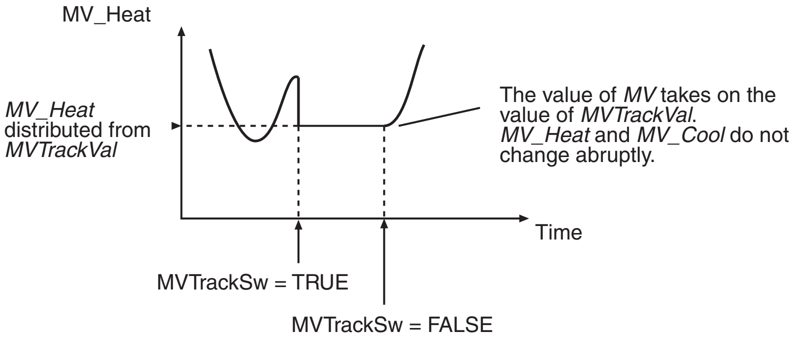
This instruction does not use this variable. Any value that is set is ignored.

● *MVTrackSw* (MV Tracking Switch)

MV tracking is a function that sets the *MV* to an external input value (called the MV tracking value) during automatic operation.

MV tracking is performed while the value of *MVTrackSw* is TRUE.

When the value of *MVTrackSw* changes to FALSE, the value of *MV* returns to the result of heating/cooling PID processing. At this time, the value of *MV* takes on the value of *MVTrackVal*. This prevents the values of *MV_Heat* and *MV_Cool* from changing abruptly.



● *MVTrackVal* (MV Tracking Value)

This is the value to which *MV* is set during MV tracking. The value of *MVTrackVal* is limited by the values of *MVLowLmt* and *MVUpLmt*.

● *StopMV* (Stop MV)

This is the value to which *MV* is set when the value of *Run* is FALSE (i.e., when execution of this instruction is stopped).

● *ErrorMV* (Error MV)

This is the value to which *MV* is set when an error occurs (i.e., when the value of *Error* is TRUE). If the value of *ErrorMV* is not within the valid range (–320 to 320), the value of *MV* will be 0 when an error occurs.

● *Alpha* (2-PID Parameter α)

This parameter determines the coefficient of the set point filter.

Refer to the description in 2-PID Control with Set Point Filter in the section on the PIDAT instruction (page 2-670) for details.

Normally set the value of *Alpha* to 0.65.

● *ATCalcGain* (Autotuning Calculation Gain)

This variable gives the coefficient of the PID constants that were calculated by autotuning when they are applied to the actual PID constants. If a value of 1.00 is specified, the results of autotuning are used directly. Increase the value of *ATCalcGain* to give priority to stability and decrease it to give priority to response.

● *ATHystrs* (Autotuning Hysteresis)

This is the hysteresis that is used in the limit cycle for autotuning. More accurate tuning is achieved if the value of *ATHystrs* is small. However, if the process value is not stable and proper autotuning is difficult, increase the value.

Refer to the description of autotuning in the section on the PIDAT instruction (page 2-670) for details.

- ***SampTime* (Sampling Period)**

This is the minimum value of the period for heating/cooling PID processing.

Refer to the description of the execution timing of heating/cooling PID processing for details. Heating/cooling PID processing is not performed again until the time specified for *SampTime* has elapsed since the last time heating/cooling PID processing was performed.

- ***RngLowLmt* (Lower Limit of Input Range) and *RngUpLmt* (Upper Limit of Input Range)**

These are the lower limit and upper limit of *PV* and *SP*. An error will occur if the value of the parameter connected to *PV* or *SP* exceeds either of these limits. *RngLowLmt* must always be less than *RngUpLmt*.

- ***DirOpr* (Action Direction)**

This instruction does not use this variable. Any value that is set is ignored.

- ***CtlPrd_Cool* (Control Period)**

This variable sets the control period for time-proportional output of *MV_Cool* when you use this instruction together with the TimeProportionalOut instruction (page 2-733). Set the same value here and for control period *CtlPrd* of the TimeProportionalOut instruction.

If you do not use time-proportional output for *MV_Cool*, set the default value, T#20s.

- ***ProportionalBand_Heat* and *ProportionalBand_Cool* (Proportional Bands)**

This is one of the three PID constants. Refer to the description of the proportional action in the section on the PIDAT instruction (page 2-670) for details.

If the values of *ProportionalBand_Heat* and *ProportionalBand_Cool* are large, the offset will be large. Hunting occurs if a proportional band is too small.

- ***IntegrationTime_Heat* and *IntegrationTime_Cool* (Integration Times)**

This is one of the three PID constants. Refer to the description of the integral action in the section on the PIDAT instruction (page 2-670) for details.

The larger the value of *IntegrationTime_Heat* or *IntegrationTime_Cool* is, the weaker the integral action is.

- ***DerivativeTime_Heat* and *DerivativeTime_Cool* (Derivative Times)**

This is one of the three PID constants. Refer to the description of the derivative action in the section on the PIDAT instruction (page 2-670) for details.

The larger the value of *DerivativeTime_Heat* or *DerivativeTime_Cool* is, the stronger the derivative action is.

- ***ManMV* (Manual Manipulated Variable)**

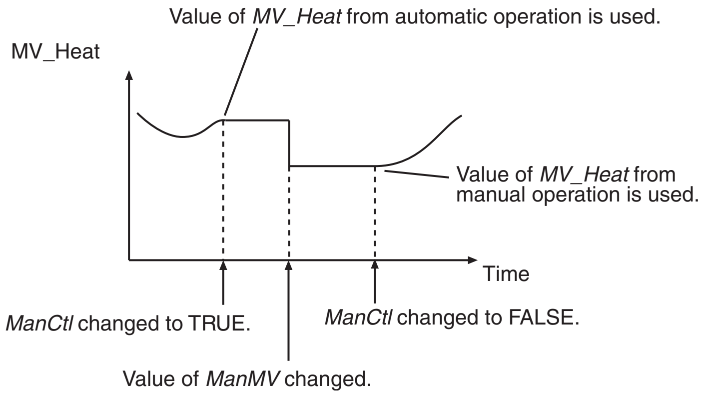
MV is set to this value during manual operation (while *ManCtl* is TRUE).

However, *MV* is set to the value of *ManMV* only when it changes after operation switches to manual operation.

The value of *MV* immediately after changing from automatic to manual operation will be the value of *MV_Heat* if that value is positive and the value of *MV_Cool* otherwise.

Also, the value of *MV* immediately after changing from manual to automatic operation will be the value of *MV_Heat* if that value is positive and the value of *MV_Cool* otherwise.

The value of *ManMV* does not have to be between *MVLowLmt* and *MVUpLmt*.



- ***ATDone* (Autotuning Normal Completion)**

This flag indicates when autotuning was completed normally. It changes to TRUE when autotuning is completed normally and remains TRUE as long as the value of *StartAT* is TRUE. It is FALSE in the following cases.

- An autotuning error end occurred.
- Autotuning is in progress (i.e., while the value of *ATBusy* is TRUE).
- Heating/cooling PID control is in progress without autotuning.
- Heating/cooling PID control is not in progress (i.e., the value of *Run* is FALSE).
- The value of *StartAT* is FALSE.

- ***ATBusy* (Autotuning Busy)**

This flag indicates when autotuning is in progress. It is TRUE while autotuning is in progress. Otherwise it is FALSE.

- ***MV* (Manipulated Variable)**

This is the manipulated variable found by the heating/cooling PID processing. *MV_Heat* and *MV_Cool* are found by distributing *MV*.

- ***MV_Heat* (Manipulated Variable for Heating Control)**

This is the manipulated variable that is applied to the heating device.

- ***MV_Cool* (Manipulated Variable for Cooling Control)**

This is the manipulated variable that is applied to the cooling device.

Heating/Cooling PID Processing

Refer to the section on the PIDAT instruction (page 2-670) for details on PID processing.

Heating/cooling PID processing is used to find the manipulated variables using the PID constants for heating control and the PID constants for cooling control. If *MV* is less than or equal to 0 in the previous processing results, the PID constants for heating control are used. If the previous *MV* is greater than 0, the PID constants for cooling control are used.

Proportional (P), Integral (I), and Derivative (D) Actions

Refer to the section on the PIDAT instruction (page 2-670) for details on the proportional action (P), integral action (I), and derivative action (D).

2-PID Control with Set Point Filter

Refer to the description in 2-PID Control with Set Point Filter in the section on the PIDAT instruction (page 2-670) for details.

Heating/Cooling PID with Autotuning

You must use the optimum PID constants to execute this instruction. There are the following two ways to achieve this.

● When Optimum PID Constants Are Not Known

If you do not know the optimum PID constants, perform autotuning at the start of operation to find them. Change the value of *Run* to TRUE while the value of *StartAT* is TRUE. First, autotuning is executed, and then heating/cooling PID control is started with the PID constants that are found.

● When Optimum PID Constants Are Known

Set *ProportionalBand_Heat*, *IntegrationTime_Heat*, *DerivativeTime_Heat*, *ProportionalBand_Cool*, *IntegrationTime_Cool*, and *DerivativeTime_Cool* to the optimum PID constants and then change the value of *Run* to TRUE.

ProportionalBand_Heat, *IntegrationTime_Heat*, *DerivativeTime_Heat*, *ProportionalBand_Cool*, *IntegrationTime_Cool*, and *DerivativeTime_Cool* are in-out variables. You cannot set constants for the input parameters. Always define suitable variables, and then assign the values to input parameters.

You can change the PID constants during operation. You can also perform autotuning during operation. To start autotuning during operation, change the value of *StartAT* to TRUE.

Control Status and Manipulated Variable

Manipulated variable *MV* is determined according to the control status as shown in the following table.

<table>
<tr><th rowspan="2">Control status</th><th colspan="5">Value of variable</th><th rowspan="2">Manipulated variable MV</th></tr>
<tr><th>ManCtl (manual/auto control)</th><th>Run (execution condition)</th><th>Error (error end)</th><th>MVTrackSw (MV tracking switch)</th><th>ATBusy (autotuning busy)</th></tr>
<tr><td>Error end</td><td rowspan="5">FALSE</td><td rowspan="4">TRUE</td><td>TRUE</td><td>---</td><td rowspan="2">FALSE</td><td>ErrorMV (error MV)</td></tr>
<tr><td>MV tracking during automatic operation</td><td rowspan="3">FALSE</td><td>TRUE</td><td>MVTrackVal (MV tracking value)</td></tr>
<tr><td>Autotuning during automatic operation</td><td rowspan="2">FALSE</td><td>TRUE</td><td>Value repeatedly changes between upper limit of MV and lower limit of MV.</td></tr>
<tr><td>Not autotuning during automatic operation</td><td rowspan="3">FALSE</td><td>Value calculated with current PID constants.</td></tr>
<tr><td>Instruction execution stopped</td><td>FALSE</td><td rowspan="2">---</td><td rowspan="2">---</td><td>StopMV (Stop MV)*1</td></tr>
<tr><td>Manual operation</td><td>TRUE</td><td>---</td><td>ManMV (manual manipulated variable)*2</td></tr>
</table>

*1 If the value of *StopMV* is outside of the valid range, the value of *MV* is 0.

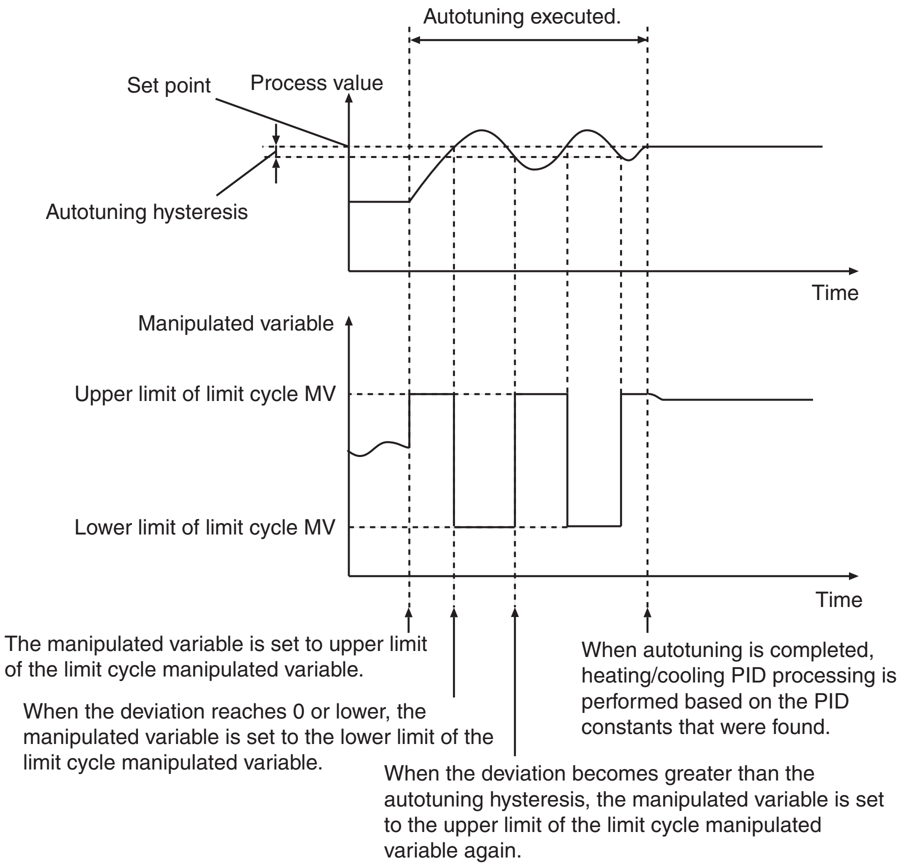
*2 If the value of *ManMV* is outside of the valid range, the value of *MV* is 0.

Autotuning

The 2-PID parameter α is not adjusted very often, so the main parameters that are adjusted for this instruction are the PID constants.

The PIDAT instruction supports autotuning of the PID constants. The limit cycle method is used for autotuning. With the limit cycle method, the manipulated variable is temporarily changed to the upper and lower limits of the limit cycle manipulated variable to find the optimum PID constants based on the resulting changes in the process value.

When you start execution of autotuning, the manipulated variable is first set to the upper limit of the limit cycle manipulated variable. When the deviation reaches 0 or lower, the manipulated variable is set to the lower limit of the limit cycle manipulated variable. When the deviation becomes greater than the autotuning hysteresis, the manipulated variable is set to the upper limit of the limit cycle manipulated variable again. This process is repeated two and a half times to calculate the optimum PID constants. The upper and lower limits of the limit cycle manipulated variable are calculated from the values of the parameters.



Autotuning is executed during heating/cooling PID control (i.e., when the value of *Run* is TRUE) if the value of *StartAT* changes to TRUE. If *StartAT* is TRUE when *Run* changes to TRUE, autotuning is executed at the start of PID control. When autotuning is completed normally, the calculated PID constants are used immediately. Autotuning is canceled if the value of *StartAT* changes to FALSE during autotuning (i.e., when *ATBusy* is TRUE). If autotuning is canceled, heating/cooling PID control is started again with the previous PID constants.

Execution Timing of Heating/Cooling PID Control

Heating/cooling PID control is repeated periodically. Heating/cooling PID processing is performed when the PIDAT instruction is executed in the user program.

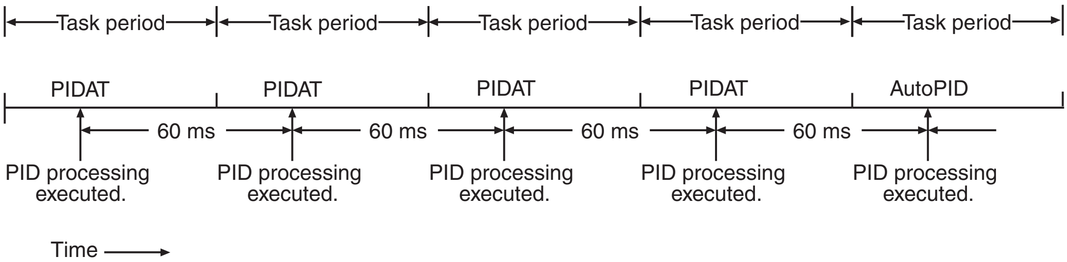
However, if sampling period *SampTime* has not elapsed since the last time heating/cooling PID processing was performed, heating/cooling PID processing is not performed.

If the elapsed time since the last time heating/cooling PID processing was executed exceeds *SampTime*, the excess time (elapsed time − *SampTime*) is carried forward to the next period.

Even if this instruction is not executed as a result of the PrgStop or MC instruction, the elapsed time from the last execution of heating/cooling PID processing is set to 0 at the timing shown by “PID processing executed” in the following figures.

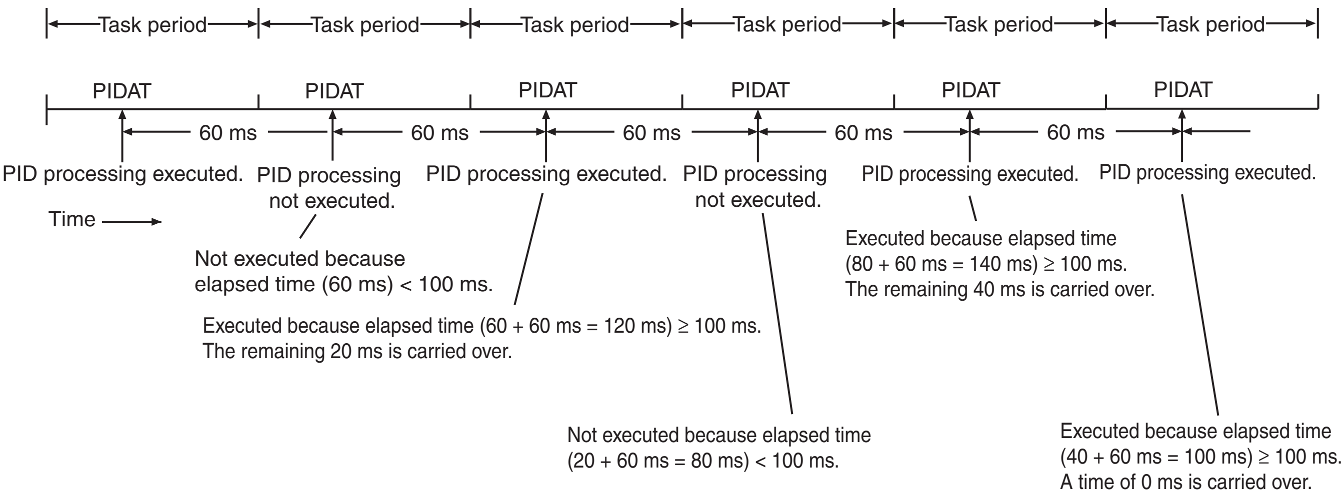
Task period = 60 ms and *SampTime* < 60 ms

The task period is greater than or equal to *SampTime*, so PID processing is executed once every task period.



Task period = 60 ms and *SampTime* = 100 ms

The task period is less than *SampTime*, so DIP processing is not executed every period.

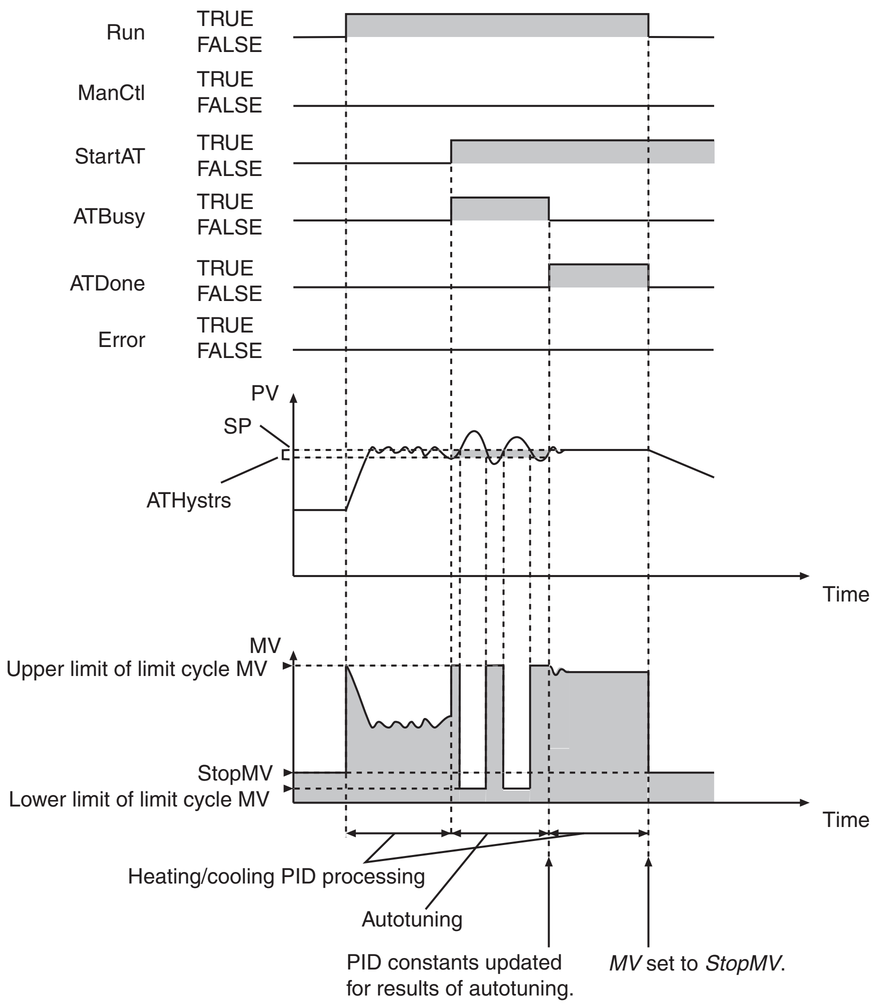


Timing Charts

Timing charts for the instruction variables are provided below for different situations.

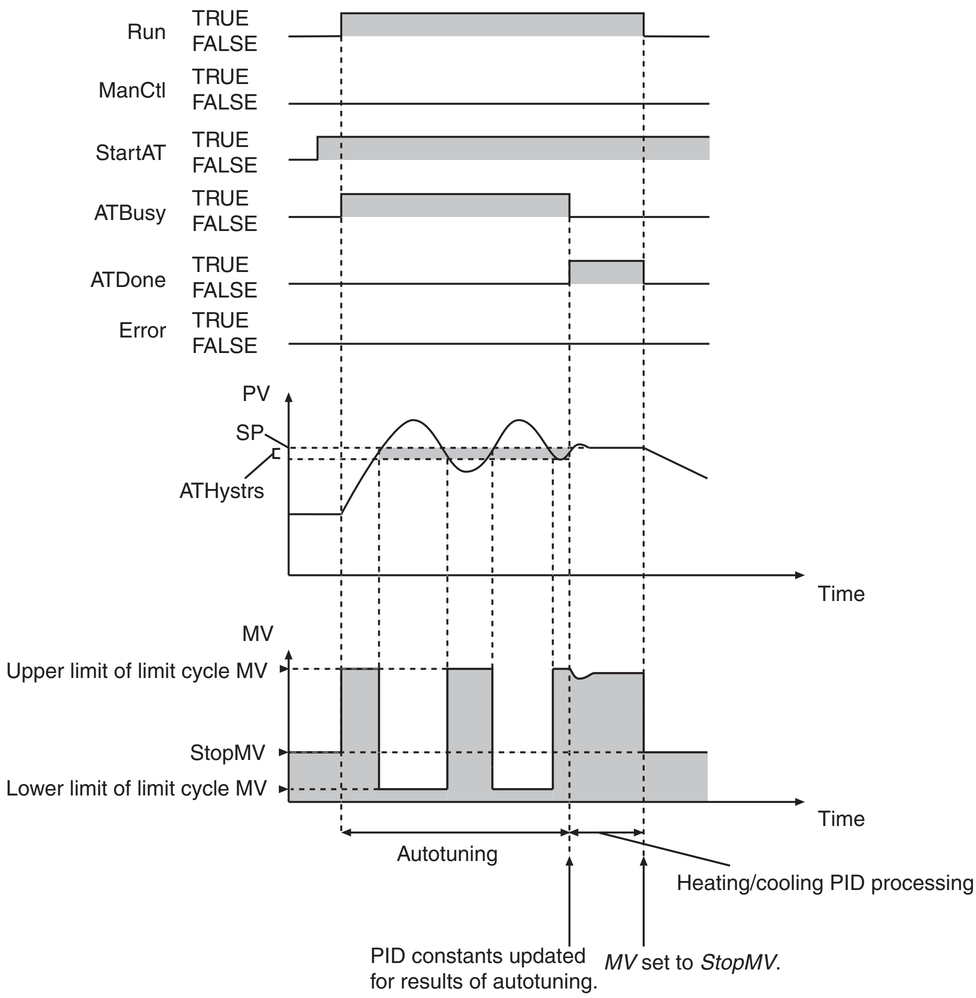
● Autotuning Executed during Automatic Operation

- In the following figure, the value of *ManCtl* is FALSE, so the value of *MV* will be *StopMV* as long as the value of *Run* is FALSE.
- When the value of *Run* changes to TRUE, *MV* is output based on the PID constants.
- Autotuning is executed when the value of *StartAT* changes to TRUE. The value of *ATBusy* changes to TRUE.
- When autotuning is completed, the value of *ATBusy* changes to FALSE and the value of *ATDone* changes to TRUE.
- After autotuning is completed, *MV* is output based on the PID constants that were found with autotuning.
- When the value of *Run* changes to FALSE, the value of *MV* changes to *StopMV*. Also, the value of *ATDone* changes to FALSE.



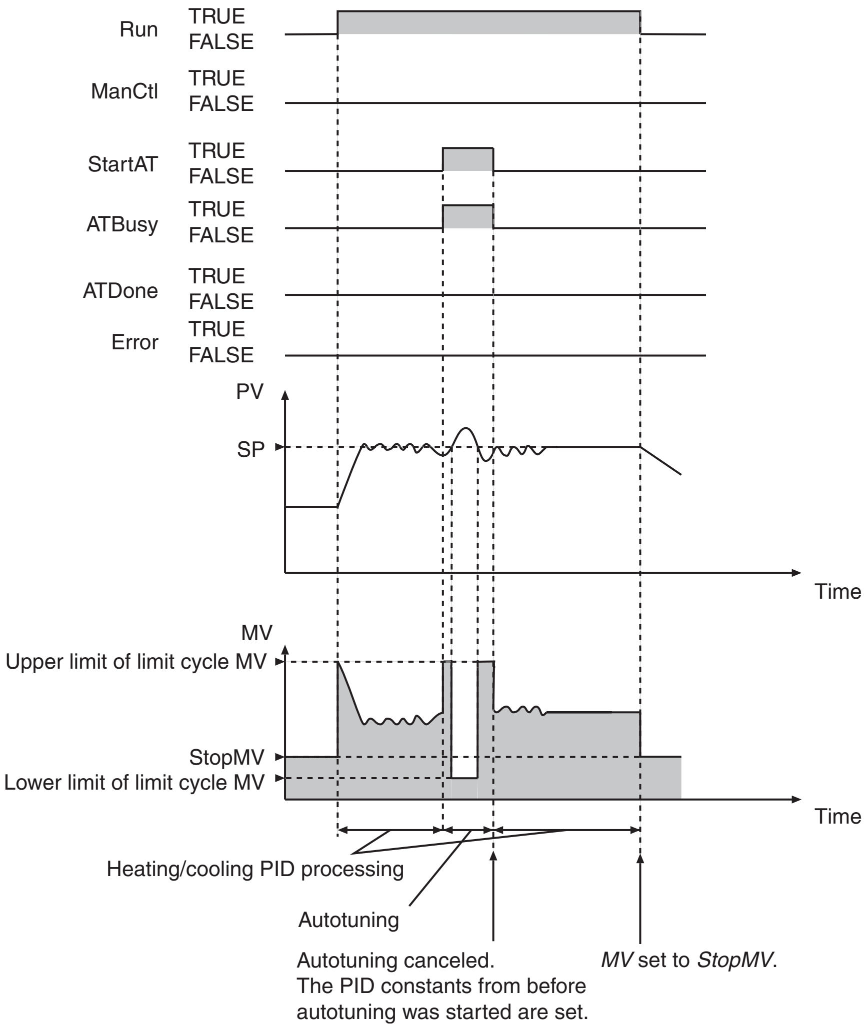
Autotuning Executed at the Start of PIDAT Execution

- In the following figure, the value of *ManCtl* is FALSE, so the value of *MV* will be *StopMV* as long as the value of *Run* is FALSE.
- While the value of *Run* is TRUE, autotuning is not executed even if the value of *StartAT* changes to TRUE.
- Autotuning is executed when the values of both *StartAT* and *Run* change to TRUE. The value of *ATBusy* changes to TRUE.
- When autotuning is completed, the value of *ATBusy* changes to FALSE and the value of *ATDone* changes to TRUE.
- After autotuning is completed, *MV* is output based on the PID constants that were found with autotuning.



● Autotuning Canceled

- In the following figure, the value of *ManCtl* is FALSE, so the value of *MV* will be *StopMV* as long as the value of *Run* is FALSE.
- When the value of *Run* changes to TRUE, *MV* is output based on the PID constants.
- Autotuning is executed when the value of *StartAT* changes to TRUE. The value of *ATBusy* changes to TRUE.
- Autotuning is canceled if the value of *StartAT* changes to FALSE during autotuning. The value of *ATBusy* changes to FALSE.
- After autotuning is completed, *MV* is output based on the PID constants from just before autotuning was started.
- When the value of *Run* changes to FALSE, the value of *MV* changes to *StopMV*.
- The value of *ATDone* does not change to TRUE because autotuning was aborted.



● An Autotuning Error Occurs during Autotuning

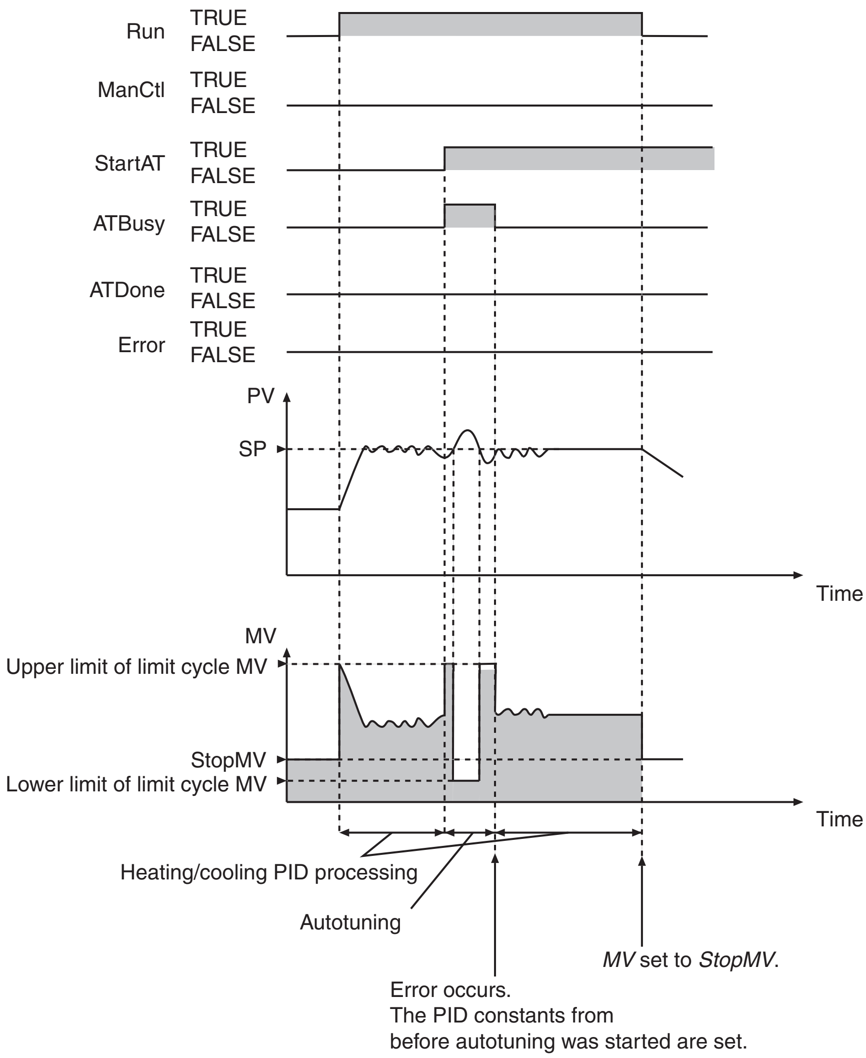
An autotuning error occurs and autotuning is stopped in the following cases.

- If the manipulated variable equals the upper limit of the limit cycle manipulated variable and the time for the deviation to reach 0 exceeds 19,999 s.
- If the manipulated variable equals the lower limit of the limit cycle manipulated variable and the time for the deviation to reach *ATHystrs* or higher exceeds 19,999 s.

The value of *Error* does not change to TRUE even if an error occurs during autotuning. Autotuning is also not recorded in the event log.

If autotuning is canceled, heating/cooling PID control is started again with the previous PID constants.

- In the following figure, the value of *ManCtl* is FALSE, so the value of *MV* will be *StopMV* as long as the value of *Run* is FALSE.
- When the value of *Run* changes to TRUE, *MV* is output based on the PID constants.
- Autotuning is executed when the value of *StartAT* changes to TRUE. The value of *ATBusy* changes to TRUE.
- Autotuning is canceled immediately if an autotuning error occurs during execution of autotuning. The value of *ATBusy* changes to FALSE.
- The value of *Error* does not change to TRUE even if an error occurs during autotuning.
- After autotuning is canceled, *MV* is output based on the PID constants from just before autotuning was started.
- When the value of *Run* changes to FALSE, the value of *MV* changes to *StopMV*.
- The value of *ATDone* does not change to TRUE because autotuning was aborted.



Additional Information

Adjusting PID Constants

Refer to the section on the PIDAT instruction (page 2-670) for the adjustment methods for PID constants.

Initial PID Constants for Temperature Control

If you use the PIDAT instruction for temperature control, use the following initial values of the PID constants as reference. Use the default values for the other variables.

Variables	Initial values (reference values)*1
ProportionalBand_Heat and ProportionalBand_Cool	10%FS
IntegrationTime_Heat and IntegrationTime_Cool	233 s
DerivativeTime_Heat and DerivativeTime_Cool	40 s

*1 If you perform autotuning, use the results from autotuning.

Precautions for Correct Use

- The values of PV and SP must be between the values of *RngLowLmt* and *RngUpLmt*, inclusive. Align the units of these variables as shown below.

Unit	Values of *PV* and *SP*	Values of *RngLowLmt* and *RngUpLmt*
% FS	PV = (Process value in physical units – MIN)/(MAX – MIN) × 100 SP = (Set point in physical units – MIN)/(MAX – MIN) × 100*1	RngLowLmt = 0 RngUpLmt = 100
Physical unit	PV = Process value in physical units SV = Set point in physical units	RngLowLmt = MIN RngUpLmt = MAX*1

*1 MAX: Upper limit of input range in physical units, MIN: Lower limit of input range in physical units,

- The following table shows which variables can be changed depending on the operating status.

Variables	Control status		
	Instruction execution stopped*1	**Automatic operation when autotuning is not being executed*2**	**Automatic operation when autotuning is being executed*3**
Run	Possible	Possible	Possible
ManCtl	Possible	Possible	Possible
StartAT	Possible	Possible	Possible
DeadBand	Possible	Possible	Possible
PV	Possible	Possible	Possible
SP	Possible	Possible	Not possible*4
MVLowLmt	Possible	Possible	Not possible*4
MVUpLmt	Possible	Possible	Not possible*4
ManResetVal*5	---	---	---
MVTrackSw	Possible	Possible	Not possible*4
MVTrackVal	Possible	Possible	Not possible*4
StopMV	Possible	Possible	Possible
ErrorMV	Possible	Possible	Possible
Alpha	Possible	Possible	Not possible*4
ATCalcGain	Possible	Possible	Not possible*4
ATHystrs	Possible	Possible	Not possible*4
CtlPrdCool	Possible	Possible	Not possible*4
SampTime	Possible	Not possible*6	Not possible*4
RngLowLmt	Possible	Not possible*6	Not possible*4
RngUpLmt	Possible	Not possible*6	Not possible*4
DirOpr*5	---	---	---
ProportionalBand_Heat	Possible	Possible	Not possible*7
IntegrationTime_Heat	Possible	Possible	Not possible*7
DerivativeTime_Heat	Possible	Possible	Not possible*7
ProportionalBand_Cool	Possible	Possible	Not possible*7
IntegrationTime_Cool	Possible	Possible	Not possible*7
DerivativeTime_Cool	Possible	Possible	Not possible*7
ManMV	Possible	Possible	Possible

*1 *ManCtl* is TRUE, *Run* is FALSE, *Error* is TRUE, or *MVTrackSw* is TRUE.

*2 *ManCtl* is FALSE, Run is TRUE, Error is FALSE, MVTrackSw is FALSE, and ATBusy is FALSE.

*3 *ManCtl* is FALSE, Run is TRUE, Error is FALSE, MVTrackSw is FALSE, and ATBusy is TRUE.

*4 Autotuning is executed with the value from just before execution of autotuning.

*5 This instruction does not use this variable. You can change the value, but it is ignored.

*6 Operation is performed with the value from just before the execution of the operation.

*7 You can change the value, but it is ignored. When autotuning is completed, the values are overwritten with the values calculated with autotuning.

- *SampTime* is truncated below 100 nanoseconds.
- If the value of *StartAT* changes to TRUE while the value of *ManCtl* is TRUE, autotuning starts the next time the value of *ManCtl* changes to FALSE.
- If the value of *ErrorMV* is not within the valid range (–320 to 320), the value of *MV* will be 0 when an error occurs.
- Autotuning is canceled if the value of *ManCtl* changes to TRUE during autotuning.
- The value of *Error* does not change to TRUE even if an error occurs during autotuning. Autotuning is also not recorded in the event log.
- An error occurs in the following case. *Error* will change to TRUE, and an error code is assigned to *ErrorID*. *ATDone* and *ATBusy* change to FALSE. *MV* is set to the value of *ErrorMV* if the values of *ManCtl* and *Run* are FALSE. If the value of *ErrorMV* is outside of the valid range, the value of *MV* is 0.

Error	Value of *ErrorID*
The value of an input variable is outside of the valid range.	16#0400
RngLowLmt is greater than or equal to *RngUpLmt*.	16#0401
MVLowLmt is greater than or equal to *MVUpLmt*.	

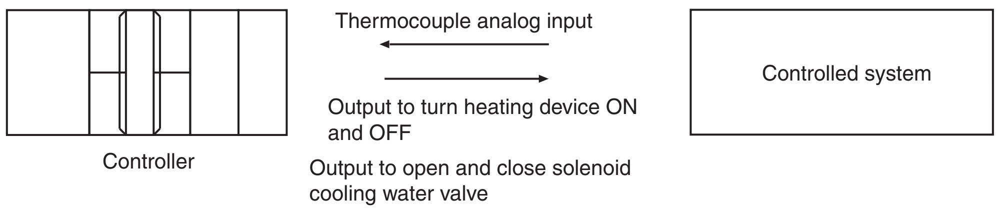
- If an error stop is required for conditions other than the above, program the system so that the value of *Run* changes to FALSE when the error occurs.
- If an error occurs because the value of *PV* or *SP* exceeds the valid range, the error status is maintained for five seconds even if the value returns to within the valid range sooner. That is, the value of *Error* will remain FALSE for five seconds.
- Heating/cooling PID control is restarted automatically if the value of *Run* is TRUE after the error is reset. Autotuning is restarted automatically if the values of *Run* and *StartAT* are TRUE.
- A check is made for errors each sampling period.
- If backup and restore operations are performed under the following conditions, the PID constants that were found with autotuning will revert to the values from before the backup operation. Use it with caution.
  - A Retain attribute is specified for the in-out parameters.
  - The operations are performed in the following order: backup, autotuning, and then restore.
- When you change from automatic operation to manual operation, the value of *MV_Heat* or *MV_Cool*, whichever is positive, is taken on to achieve bumpless operation (i.e., to prevent abrupt changes). Therefore, the value of the other variable may change abruptly.

Version Information

A CPU Unit with unit version 1.08 or later and Sysmac Studio version 1.09 or higher are required to use this instruction.

Sample Programming

In this sample, the PIDAT_HeatCool instruction is used to perform temperature control. There is one analog thermocouple input from the controlled system. There are two outputs to the controlled system, a heating digital output and a cooling digital output. The heating digital output turns the heating device ON and OFF. The cooling digital output opens and closes the solenoid valve for the cooling water.



Unit Configuration

The following Units are connected.

- CJ1W-AD04U Isolated-type Universal Input Unit
- CJ1W-OC201 Relay Contact Output Unit

I/O Map

The I/O maps for the Units are set as shown in the following tables.

● C1JW-AD04U

Port	Description	Read/ write	Data type	Variable	Variable comment	Variable type
Ch1_AllnPV	Process value for input 1	R	INT	J01_Ch1_AllnPV	Thermocouple input	Global variable

● CJ1W-OC201

Port	Description	Read/ write	Data type	Variable	Variable comment	Variable type
Ch1_Out00	Bit 00 of output word 1	RW	BOOL	J02_Ch1_Out00	Output to heating device	Global variable
Ch1_Out04	Bit 04 of output word 1	RW	BOOL	J02_Ch1_Out04	Output to cooling device	Global variable

Touch Panel Specifications

This sample assumes that a touch panel is connected to the Controller. The following I/O information is handled through the touch panel.

I/O	Information
Inputs	Sample programming execution flag Manual/auto control flag Set point Autotuning execution flag Deadband Initial setting parameters Operation setting parameters
I/O	Proportional band, integration time, and derivative time for heating control Proportional band, integration time, and derivative time for cooling control Manual manipulated variable
Outputs	Process value Autotuning normal completion flag Autotuning executing flag Error flag Manipulated variable Manipulated variable for heating control Manipulated variable for cooling control

Converting the Manipulated Variables to Time-proportional Outputs

In this sample, a digital ON/OFF output is used for both the heating device and the cooling device. Therefore, it is necessary to convert the manipulated variables for the heating and cooling devices to time-proportional outputs. The TimeProportionalOut instruction (page 2-733) converts a manipulated variable to a time-proportional output.

However, during autotuning, the outputs to the heating and cooling devices must be changed immediately after the *MV_Heat* and *MV_Cool* outputs from the PIDAT_HeatCool instruction change. Therefore, the TimeProportionalOut instruction cannot be used. If the TimeProportionalOut instruction was used, the outputs to the heating and cooling devices would change only at the control period that was set by the user. In this sample, timer instructions are used to convert the manipulated variables to time-proportional outputs during autotuning.

Application Programming

Definitions of Global Variables

Global Variables

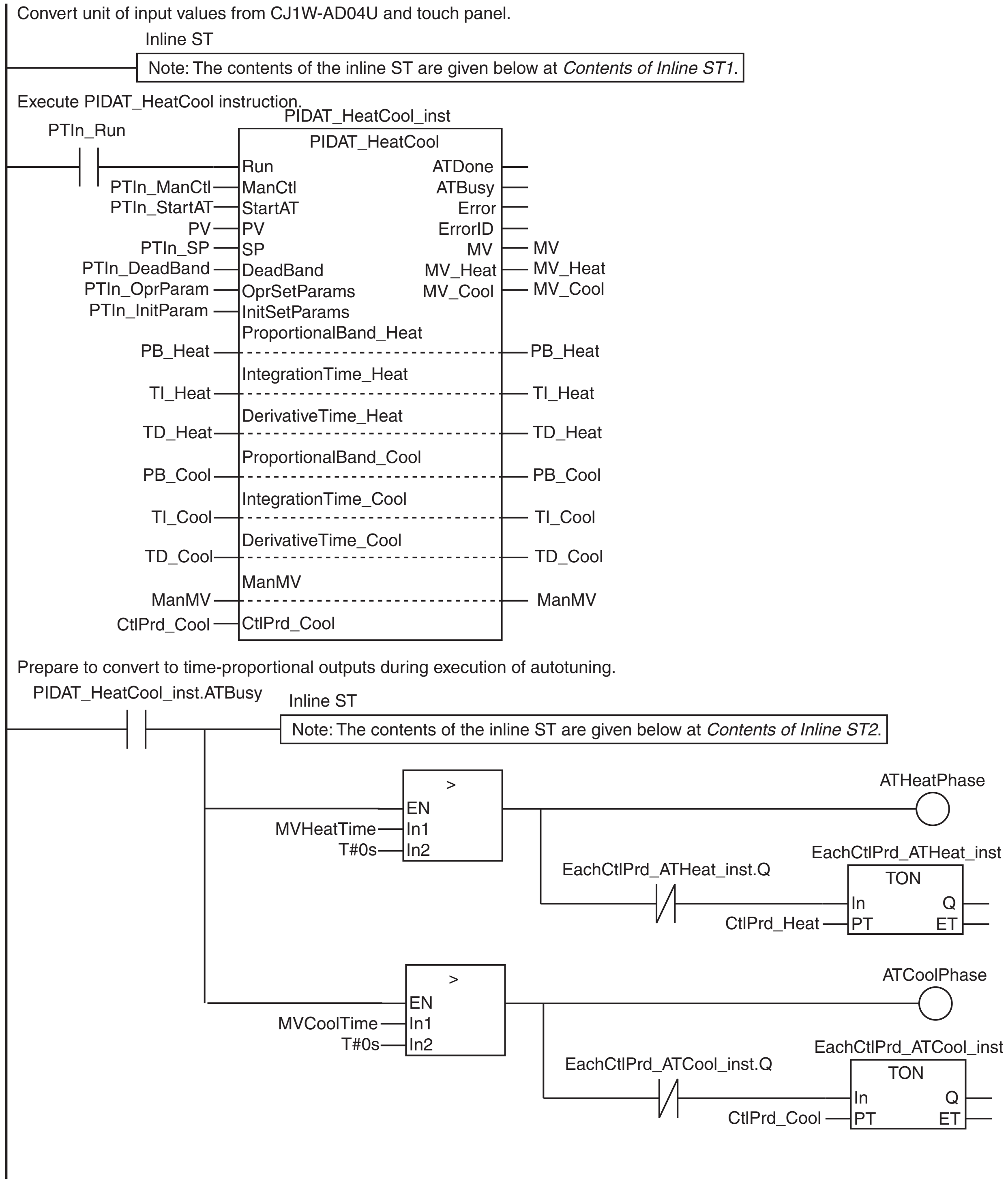
Variable	Data type	Initial value	AT	Retain	Network Publish	Comment
J01_Ch1_AI-InPV	INT	0	IOBus://rack#0/slot#0/Ch1_AIInPV		Not published.	Thermocouple input from CJ1W-AD04U
J02_Ch1_Out00	BOOL	FALSE	IOBus://rack#0/slot#1/Ch1_Out/Ch1_Out00		Not published.	Heating output to CJ1W-OC201
J02_Ch1_Out04	BOOL	FALSE	IOBus://rack#0/slot#1/Ch1_Out/Ch1_Out04		Not published.	Cooling output to CJ1W-OC201
PTIn_Run	BOOL	FALSE		✔	Input	Sample programming execution flag input from touch panel
PTIn_ManCtl	BOOL	FALSE		✔	Input	Manual/auto control flag input from touch panel
PTIn_SP	REAL			✔	Input	Set point input from touch panel
PTIn_StartAT	BOOL	FALSE		✔	Input	Autotuning execution flag input from touch panel
PTIn_Dead-Band	REAL	0		✔	Input	Deadband input from touch panel
PTIn_Init-Param	_sIN-IT_SET_PARAMS	(SampTime := T#100ms, RngLowLmt := 0.0, RngUpLmt := 100.0, DirOpr := False)		✔	Input	Initial setting parameter input from touch panel
PTIn_InitSe-tOpr_Samp-Time	LINT	100		✔	Input	Sampling period input from touch panel (unit: ms)
PTIn_Opr-Param	_sO-PR_SET_PARAMS	(MVLowLmt := -100, MVUpLmt := 100, ManResetVal := 0.0, MVTrackSw := False, MVTrackVal := 0.0, StopMV := 0.0, ErrorMV := 0.0, Alpha := 0.65, ATCalcGain := 1.0, ATHystrs := 0.2)		✔	Input	Operation setting parameter input from touch panel
PTOut_PV	REAL	0			Output	Process value output to touch panel
PT_PB_Heat	REAL	1		✔	Input	Proportional band for heating control I/O from touch panel

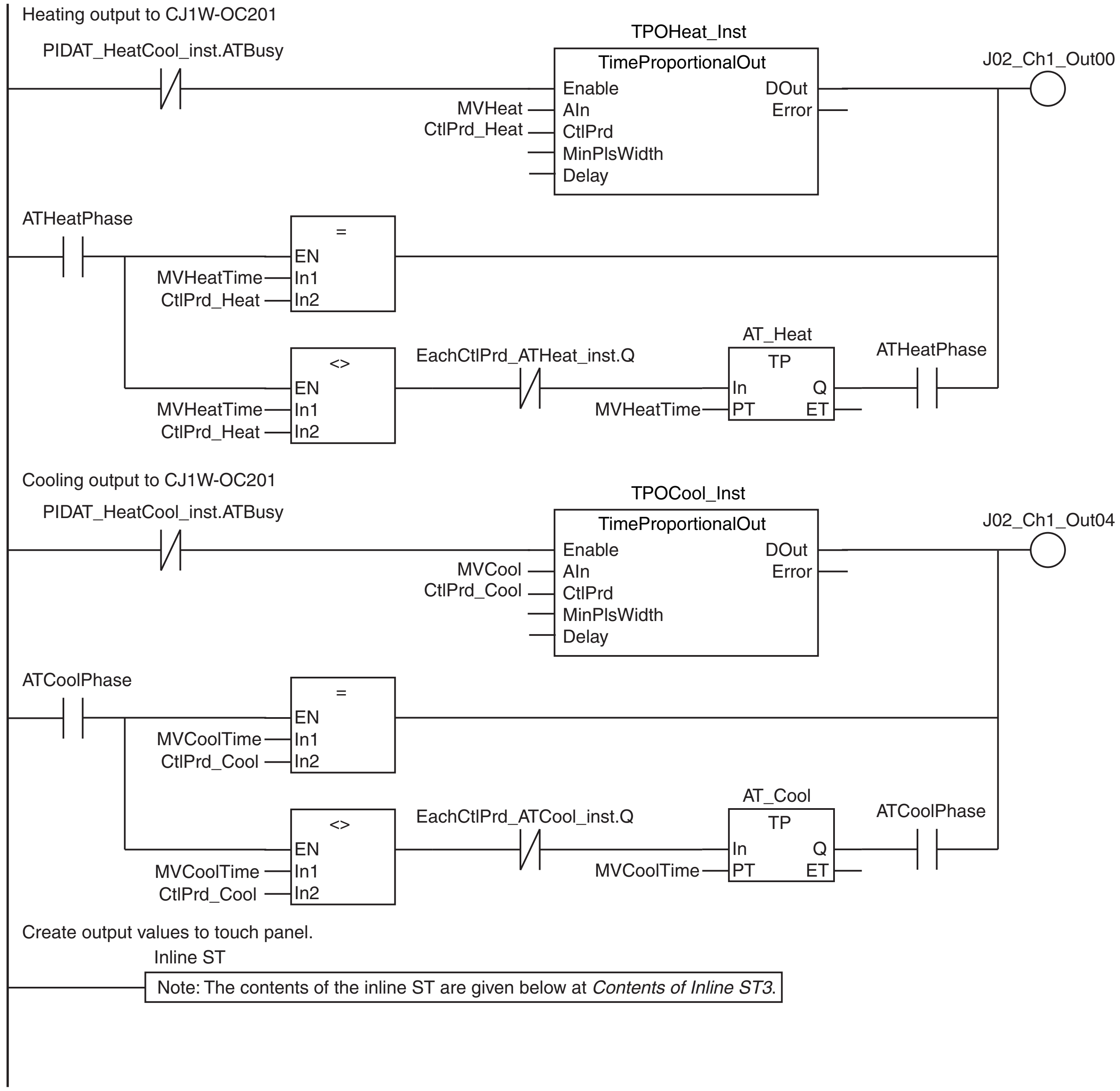
Variable	Data type	Initial value	AT	Retain	Network Publish	Comment
PT_TI_Heat	LINT	1000		✔	Input	Integration time for heating control I/O from touch panel (unit: ms)
PT_TD_Heat	LINT	1000		✔	Input	Derivative time for heating control I/O from touch panel (unit: ms)
PT_PB_Cool	REAL	1		✔	Input	Proportional band for cooling control I/O from touch panel
PT_TI_Cool	LINT	1000		✔	Input	Integration time for cooling control I/O from touch panel (unit: ms)
PT_TD_Cool	LINT	1000		✔	Input	Derivative time for cooling control I/O from touch panel (unit: ms)
PT_ManMV	REAL	0		✔	Input	Manual manipulated variable I/O from touch panel
PTOut_AT-Done	BOOL	FALSE			Output	Autotuning normal completion flag output to touch panel
PTOut_AT-Busy	BOOL	FALSE			Output	Autotuning executing flag output to touch panel
PTOut_Error	BOOL	FALSE			Output	Error flag output to touch panel
PTOut_MV	REAL	0			Output	Manipulated variable output to touch panel
PTOut_M-VHeat	REAL	0			Output	Manipulated variable for heating control output to touch panel
PTOut_M-VCool	REAL	0			Output	Manipulated variable for cooling control output to touch panel

LD

Internal Vari-ables	Variable	Data type	Initial value	Comment
	PB_Heat	REAL	0	Proportional band for heating con-trol
	PB_Cool	REAL	0	Proportional band for cooling con-trol
	MV	REAL	0	Manipulated variable
	MV_Heat	REAL	0	Manipulated variable for heating control
	MV_Cool	REAL	0	Manipulated variable for cooling control
	PIDAT_HeatCool_inst	PIDAT_HeatCool		Instance of PIDAT_HeatCool instruction
	TI_Heat	TIME	T#0s	Integration time for heating control
	TI_Cool	TIME	T#0s	Integration time for cooling control
	TD_Heat	TIME	T#0s	Derivative time for heating control
	TD_Cool	TIME	T#0s	Derivative time for cooling control
	ManMV	REAL	0	Manual manipulated variable
	CtlPrd_Cool	TIME	T#20s	Cooling control period
	CtlPrd_Heat	TIME	T#2s	Heating control period
	TPOHeat_inst	TimeProportion-alOut		Instance of TimeProportionalOut instruction for heating control
	TPOCool_inst	TimeProportion-alOut		Instance of TimeProportionalOut instruction for cooling control
	ATHeatPhase	BOOL	FALSE	Autotuning heating control flag
	ATCoolPhase	BOOL	FALSE	Autotuning cooling control flag
	MVHeatTime	TIME	T#0s	Autotuning heating control time
	MVCoolTime	TIME	T#0s	Autotuning cooling control time
	AT_Heat_inst	TP		Instance of TP instruction for heat-ing control manipulated variable output during autotuning
	AT_Cool_inst	TP		Instance of TP instruction for cool-ing control manipulated variable output during autotuning
	EachCtlPrd_ATHeat_inst	TON		Instance of TON instruction for heating control manipulated vari-able output during autotuning
	EachCtlPrd_ATCool_inst	TON		Instance of TON instruction for cooling control manipulated vari-able output during autotuning
	PV	REAL	0	Process value

External Vari-ables	Variable	Data type	Comment
	J01_Ch1_AIInPV	INT	Thermocouple input from CJ1W-AD04U
	J02_Ch1_Out00	BOOL	Heating output to CJ1W-OC201
	J02_Ch1_Out04	BOOL	Cooling output to CJ1W-OC201
	PTIn_Run	BOOL	Sample programming execution flag input from touch panel
	PTIn_ManCtl	BOOL	Manual/auto control flag input from touch panel
	PTIn_SP	REAL	Set point input from touch panel
	PTIn_StartAT	BOOL	Autotuning execution flag input from touch panel
	PTIn_DeadBand	REAL	Deadband input from touch panel
	PTIn_InitParam	_sINIT_SET_PARAMS	Initial setting parameter input from touch panel
	PTIn_InitSetOpr_SampTime	LINT	Sampling period input from touch panel (unit: ms)
	PTIn_OprParam	_sOPR_SET_PARAMS	Operation setting parameter input from touch panel
	PTOut_PV	REAL	Process value output to touch panel
	PT_PB_Heat	REAL	Proportional band for heating control I/O from touch panel
	PT_TI_Heat	LINT	Integration time for heating control I/O from touch panel (unit: ms)
	PT_TD_Heat	LINT	Derivative time for heating control I/O from touch panel (unit: ms)
	PT_PB_Cool	REAL	Proportional band for cooling control I/O from touch panel
	PT_TI_Cool	LINT	Integration time for cooling control I/O from touch panel (unit: ms)
	PT_TD_Cool	LINT	Derivative time for cooling control I/O from touch panel (unit: ms)
	PT_ManMV	REAL	Manual manipulated variable I/O from touch panel
	PTOut_ATDone	BOOL	Autotuning normal completion flag out-put to touch panel
	PTOut_ATBusy	BOOL	Autotuning executing flag output to touch panel
	PTOut_Error	BOOL	Error flag output to touch panel
	PTOut_MV	REAL	Manipulated variable output to touch panel
	PTOut_MVHeat	REAL	Manipulated variable for heating control output to touch panel
	PTOut_MVCool	REAL	Manipulated variable for cooling control output to touch panel


Convert unit of input values from CJ1W-AD04U and touch panel.
Inline ST
Note: The contents of the inline ST are given below at *Contents of Inline ST1*.
Execute PIDAT_HeatCool instruction.
PIDAT_HeatCool_inst
PTIn_Run
PIDAT_HeatCool
Run
ATDone
PTIn_ManCtl
ManCtl
ATBusy
PTIn_StartAT
StartAT
Error
PV
PV
ErrorID
PTIn_SP
SP
MV
MV
PTIn_DeadBand
DeadBand
MV_Heat
MV_Heat
PTIn_OprParam
OprSetParams
MV_Cool
MV_Cool
PTIn_InitParam
InitSetParams
ProportionalBand_Heat
PB_Heat
PB_Heat
IntegrationTime_Heat
TI_Heat
TI_Heat
DerivativeTime_Heat
TD_Heat
TD_Heat
ProportionalBand_Cool
PB_Cool
PB_Cool
IntegrationTime_Cool
TI_Cool
TI_Cool
DerivativeTime_Cool
TD_Cool
TD_Cool
ManMV
ManMV
ManMV
CtlPrd_Cool
CtlPrd_Cool
Prepare to convert to time-proportional outputs during execution of autotuning.
PIDAT_HeatCool_inst.ATBusy
Inline ST
Note: The contents of the inline ST are given below at *Contents of Inline ST2*.
>
EN
MVHeatTime
In1
T#0s
In2
ATHeatPhase
EachCtlPrd_ATHeat_inst
EachCtlPrd_ATHeat_inst.Q
TON
In
Q
CtlPrd_Heat
PT
ET
>
EN
MVCoolTime
In1
T#0s
In2
ATCoolPhase
EachCtlPrd_ATCool_inst
EachCtlPrd_ATCool_inst.Q
TON
In
Q
CtlPrd_Cool
PT
ET



● Contents of Inline ST1

```
// Convert unit of input values from CJ1W-AD04U and touch panel.
PV := INT_TO_REAL(J01_Ch1_AIInPV)/REAL#10.0;

PTIn_InitParam.SampTime := NanoSecToTime(PTIn_InitSetOpr_SampTime*1000000);

PB_Heat := PT_PB_Heat;
TI_Heat := NanoSecToTime(PT_TI_Heat*1000000);
TD_Heat := NanoSecToTime(PT_TD_Heat*1000000);
PB_Cool := PT_PB_Cool;
TI_Cool := NanoSecToTime(PT_TI_Cool*1000000);
TD_Cool := NanoSecToTime(PT_TD_Cool*1000000);

ManMV := PT_ManMV;
```

Contents of Inline ST2

```
MVHeatTime := MULTIME(CtlPrd_Heat,(MV_Heat/100));
MVCoolTime := MULTIME(CtlPrd_Cool,(MV_Cool/100));
```

Contents of Inline ST3

```
// Create output values to touch panel.
PTOut_PV := PV;

PTOut_ATDone := PIDAT_HeatCool_inst.ATDone;
PTOut_ATBusy := PIDAT_HeatCool_inst.ATBusy;
PTOut_Error  := PIDAT_HeatCool_inst.Error;

PTOut_MV     := PIDAT_HeatCool_inst.MV;
PTOut_MVHeat := PIDAT_HeatCool_inst.MV_Heat;
PTOut_MVCool := PIDAT_HeatCool_inst.MV_Cool;

PT_PB_Heat := PB_Heat;
PT_TI_Heat := TimeToNanoSec( TI_Heat )/1000000;
PT_TD_Heat := TimeToNanoSec( TD_Heat )/1000000;
PT_PB_Cool := PB_Cool;
PT_TI_Cool := TimeToNanoSec( TI_Cool )/1000000;
PT_TD_Cool := TimeToNanoSec( TD_Cool )/1000000;

PT_ManMV := ManMV;
```

ST

Internal Vari-ables	Variable	Data type	Initial value	Comment
	PB_Heat	REAL	0	Proportional band for heating con-trol
	PB_Cool	REAL	0	Proportional band for cooling con-trol
	MV	REAL	0	Manipulated variable
	MV_Heat	REAL	0	Manipulated variable for heating control
	MV_Cool	REAL	0	Manipulated variable for cooling control
	PIDAT_HeatCool_inst	PIDAT_HeatCool		Instance of PIDAT_HeatCool instruction
	TI_Heat	TIME	T#0s	Integration time for heating control
	TI_Cool	TIME	T#0s	Integration time for cooling control
	TD_Heat	TIME	T#0s	Derivative time for heating control
	TD_Cool	TIME	T#0s	Derivative time for cooling control
	ManMV	REAL	0	Manual manipulated variable
	CtlPrd_Cool	TIME	T#20s	Cooling control period
	CtlPrd_Heat	TIME	T#2s	Heating control period
	TPOHeat_inst	TimeProportion-alOut		Instance of TimeProportionalOut instruction for heating control
	TPOCool_inst	TimeProportion-alOut		Instance of TimeProportionalOut instruction for cooling control
	ATHeatPhase	BOOL	FALSE	Autotuning heating control flag
	ATCoolPhase	BOOL	FALSE	Autotuning cooling control flag
	MVHeatTime	TIME	T#0s	Autotuning heating control time
	MVCoolTime	TIME	T#0s	Autotuning cooling control time
	AT_Heat_inst	TP		Instance of TP instruction for heat-ing control manipulated variable output during autotuning
	AT_Cool_inst	TP		Instance of TP instruction for cool-ing control manipulated variable output during autotuning
	EachCtlPrd_ATHeat_inst	TON		Instance of TON instruction for heating control manipulated vari-able output during autotuning
	EachCtlPrd_ATCool_inst	TON		Instance of TON instruction for cooling control manipulated vari-able output during autotuning
	PV	REAL	0	Process value

External Vari-ables	Variable	Data type	Comment
	J01_Ch1_AIInPV	INT	Thermocouple input from CJ1W-AD04U
	J02_Ch1_Out00	BOOL	Heating output to CJ1W-OC201
	J02_Ch1_Out04	BOOL	Cooling output to CJ1W-OC201
	PTIn_Run	BOOL	Sample programming execution flag input from touch panel
	PTIn_ManCtl	BOOL	Manual/auto control flag input from touch panel
	PTIn_SP	REAL	Set point input from touch panel
	PTIn_StartAT	BOOL	Autotuning execution flag input from touch panel
	PTIn_DeadBand	REAL	Deadband input from touch panel
	PTIn_InitParam	_sINIT_SET_PARAMS	Initial setting parameter input from touch panel
	PTIn_InitSetOpr_SampTime	LINT	Sampling period input from touch panel (unit: ms)
	PTIn_OprParam	_sOPR_SET_PARAMS	Operation setting parameter input from touch panel
	PTOut_PV	REAL	Process value output to touch panel
	PT_PB_Heat	REAL	Proportional band for heating control I/O from touch panel
	PT_TI_Heat	LINT	Integration time for heating control I/O from touch panel (unit: ms)
	PT_TD_Heat	LINT	Derivative time for heating control I/O from touch panel (unit: ms)
	PT_PB_Cool	REAL	Proportional band for cooling control I/O from touch panel
	PT_TI_Cool	LINT	Integration time for cooling control I/O from touch panel (unit: ms)
	PT_TD_Cool	LINT	Derivative time for cooling control I/O from touch panel (unit: ms)
	PT_ManMV	REAL	Manual manipulated variable I/O from touch panel
	PTOut_ATDone	BOOL	Autotuning normal completion flag output to touch panel
	PTOut_ATBusy	BOOL	Autotuning executing flag output to touch panel
	PTOut_Error	BOOL	Error flag output to touch panel
	PTOut_MV	REAL	Manipulated variable output to touch panel
	PTOut_MVHeat	REAL	Manipulated variable for heating control output to touch panel
	PTOut_MVCool	REAL	Manipulated variable for cooling control output to touch panel

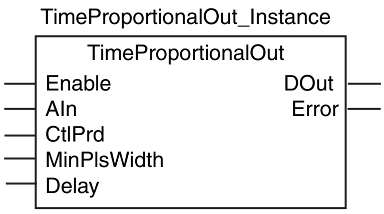
```
// Convert unit of input values from CJ1W-AD04U and touch panel.
PV := INT_TO_REAL(J01_Ch1_AIInPV)/REAL#10.0;

PTIn_InitParam.SampTime := NanoSecToTime(PTIn_InitSetOpr_SampTime*1000000);
```

```
PB_Heat := PT_PB_Heat;
TI_Heat := NanoSecToTime(PT_TI_Heat*1000000);
TD_Heat := NanoSecToTime(PT_TD_Heat*1000000);
PB_Cool := PT_PB_Cool;
TI_Cool := NanoSecToTime(PT_TI_Cool*1000000);
TD_Cool := NanoSecToTime(PT_TD_Cool*1000000);

ManMV   := PT_ManMV;

// Execute PIDAT_HeatCool instruction.
PIDAT_HeatCool_inst(Run                    :=PTIn_Run,
                    ManCtl                 :=PTIn_ManCtl,
                    StartAT                :=PTIn_StartAT,
                    PV                     :=PV,
                    SP                     :=PTIn_SP,
                    DeadBand               :=PTIn_DeadBand,
                    OprSetParams           :=PTIn_OprParam,
                    InitSetParams          :=PTIn_InitParam,
                    ProportionalBand_Heat  :=PB_Heat,
                    IntegrationTime_Heat   :=TI_Heat,
                    DerivativeTime_Heat    :=TD_Heat,
                    ProportionalBand_Cool  :=PB_Cool,
                    IntegrationTime_Cool   :=TI_Cool,
                    DerivativeTime_Cool    :=TD_Cool,
                    ManMV                  :=ManMV,
                    CtlPrd_Cool            :=CtlPrd_Cool,
                    MV                     =>MV,
                    MV_Heat                =>MV_Heat,
                    MV_Cool                =>MV_Cool);

// Prepare to convert to time-proportional outputs during execution
// of autotuning.
IF PIDAT_HeatCool_inst.ATBusy THEN
   MVHeatTime := MULTIME(CtlPrd_Heat, (MV_Heat/100) );
   MVCoolTime := MULTIME(CtlPrd_Cool, (MV_Cool/100) );
END_IF;

ATHeatPhase := PIDAT_HeatCool_inst.ATBusy & (MVHeatTime>T#0s);
EachCtlPrd_ATHeat_inst(In:= ATHeatPhase & NOT(EachCtlPrd_ATHeat_inst.Q),
                       PT:= CtlPrd_Heat);

ATCoolPhase := PIDAT_HeatCool_inst.ATBusy & (MVCoolTime>T#0s);
EachCtlPrd_ATCool_inst(In:= ATCoolPhase & NOT(EachCtlPrd_ATCool_inst.Q),
                       PT:= CtlPrd_Cool);

// Heating output to CJ1W-OC201
TPOHeat_inst(Enable :=NOT(PIDAT_HeatCool_inst.ATBusy),
             AIn    :=MV_Heat,
             CtlPrd :=CtlPrd_Heat );
AT_Heat_inst(In:= ATHeatPhase & (MVHeatTime<>CtlPrd_Heat) &
NOT(EachCtlPrd_ATHeat_inst.Q) ,
             PT:= MVHeatTime);
J02_Ch1_Out00 :=( TPOHeat_inst.DOut ) OR
                ( ATHeatPhase & (MVHeatTime=CtlPrd_Heat)) OR
                ( AT_Heat_inst.Q & ATHeatPhase );

// Cooling output to CJ1W-OC201
TPOCool_inst(Enable :=NOT(PIDAT_HeatCool_inst.ATBusy),
             AIn    :=MV_Cool,
             CtlPrd :=CtlPrd_Cool );
AT_Cool_inst(In:= ATCoolPhase & (MVCoolTime<>CtlPrd_Cool) &
NOT(EachCtlPrd_ATCool_inst.Q) ,
             PT:= MVCoolTime);
J02_Ch1_Out04 :=( TPOCool_inst.DOut ) OR
                ( ATCoolPhase & (MVCoolTime=CtlPrd_Cool)) OR
```

```
                 ( AT_Cool_inst.Q & ATCoolPhase );

// Create output values to touch panel.
PTOut_PV := PV;

PTOut_ATDone := PIDAT_HeatCool_inst.ATDone;
PTOut_ATBusy := PIDAT_HeatCool_inst.ATBusy;
PTOut_Error  := PIDAT_HeatCool_inst.Error;

PTOut_MV     := PIDAT_HeatCool_inst.MV;
PTOut_MVHeat := PIDAT_HeatCool_inst.MV_Heat;
PTOut_MVCool := PIDAT_HeatCool_inst.MV_Cool;

PT_PB_Heat := PB_Heat;
PT_TI_Heat := TimeToNanoSec(TI_Heat)/1000000;
PT_TD_Heat := TimeToNanoSec(TD_Heat)/1000000;
PT_PB_Cool := PB_Cool;
PT_TI_Cool := TimeToNanoSec(TI_Cool)/1000000;
PT_TD_Cool := TimeToNanoSec(TD_Cool)/1000000;

PT_ManMV   := ManMV;
```

TimeProportionalOut

The TimeProportionalOut instruction converts a manipulated variable to a time-proportional output.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
TimeProportionalOut	Time-proportional Output	FB		TimeProportionalOut_instance(Enable, AIn, CtlPrd, MinPlsWidth, Delay, DOut, Error);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Enable	Enable	Input	TRUE: Execute FALSE: Reset time-proportional output	Depends on data type.	---	FALSE
AIn	Manipulated variable		Manipulated variable	0 to 100	%	0
CtlPrd	Control period		Control period of time-proportional output	T#0.1s to T#100s	s	T#2s
MinPlsWidth	Minimum pulse width		Minimum pulse width	0 to 50	%	1
Delay	Delay		ON-delay time	0 to 100	%	0
DOut	Time-proportional output	Output	TRUE: Time-proportional output is ON. FALSE: Time-proportional output is OFF.	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Enable	OK																			
AIn														OK						
CtlPrd																OK				
MinPlsWidth														OK						
Delay														OK						
DOut	OK																			

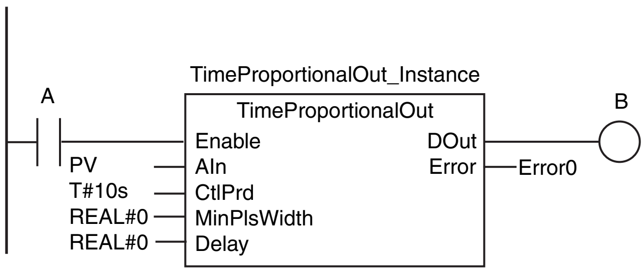
Function

The TimeProportionalOut instruction converts a manipulated variable, such as the one for PID control, to a time-proportional output. A time-proportional output converts a manipulated variable to a time ratio between ON and OFF.

While *Enable* is TRUE, the value of manipulated variable *AIn* is converted to time-proportional output *DOut* for control period *CtlPrd*. If *Enable* changes to FALSE, the time-proportional output is reset. *DOut* and *Error* change to FALSE. The values of *CtlPrd*, *MinPlsWidth*, and *Delay* are updated when *Enable* changes from FALSE to TRUE.

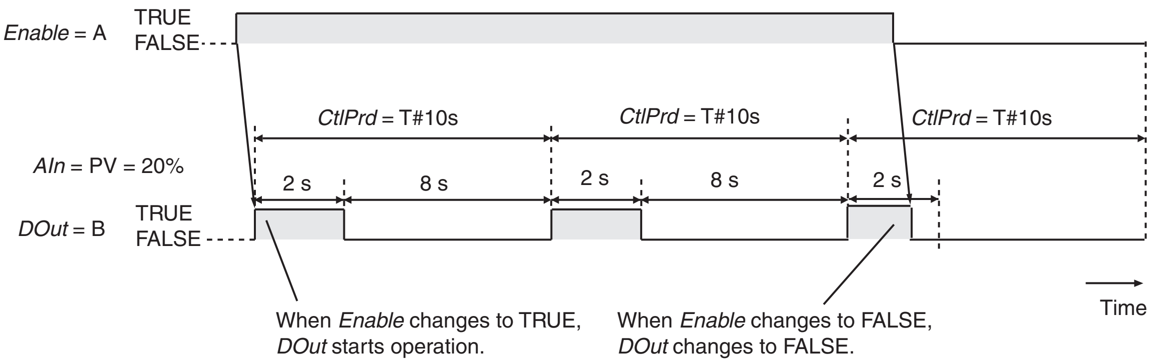
The following example is for when the value of *CtlPrd* is 10 s and the value of *AIn* is 20%. While *Enable* is TRUE, *DOut* is TRUE for two seconds and then FALSE for eight seconds. This is repeated at a 10-second period.

LD



ST

```
TimeProportionalOut_instance(A,PV,T#10s,REAL#0,REAL#0,B,Error0);
```



Resolution of Time-proportional Output *DOut*

The minimum unit for the conversion of the value of *AIn* to *DOut* is the resolution of *DOut*.

If the resolution of the value of *AIn* is higher than the resolution of *DOut*, *AIn* is rounded to the resolution of *DOut* when it is converted to *DOut*.

The resolution of *DOut* is given by the following formula.

Resolution of *DOut* (%) = Task period ÷ *CtlPrd* × 100

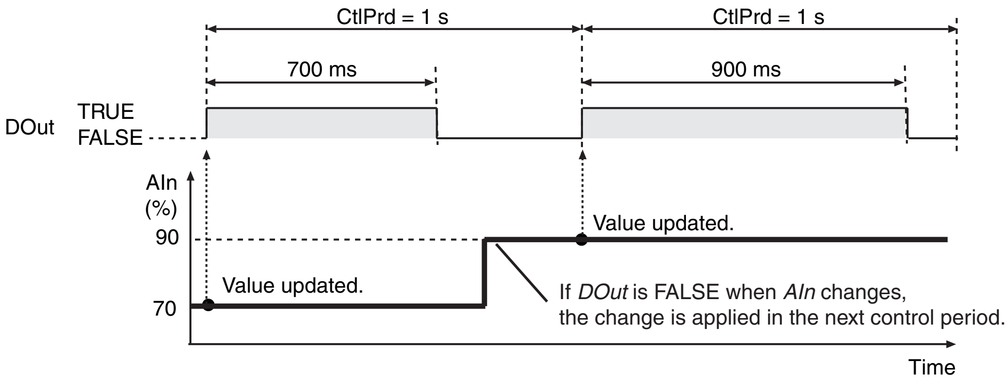
For example, if the task period is 1 ms and the value of *CtlPrd* is 1 s, the resolution of *DOut* is 0.1%. In this case, the digits after the first decimal digit of the value of *AIn* are truncated.

Update Timing of the Value of Manipulated Variable *AIn*

When value of *AIn* is updated depends on whether *DOut* is FALSE or TRUE.

● *DOut* = FALSE

While *DOut* is FALSE, any change in the value of *AIn* is applied in the next control period.

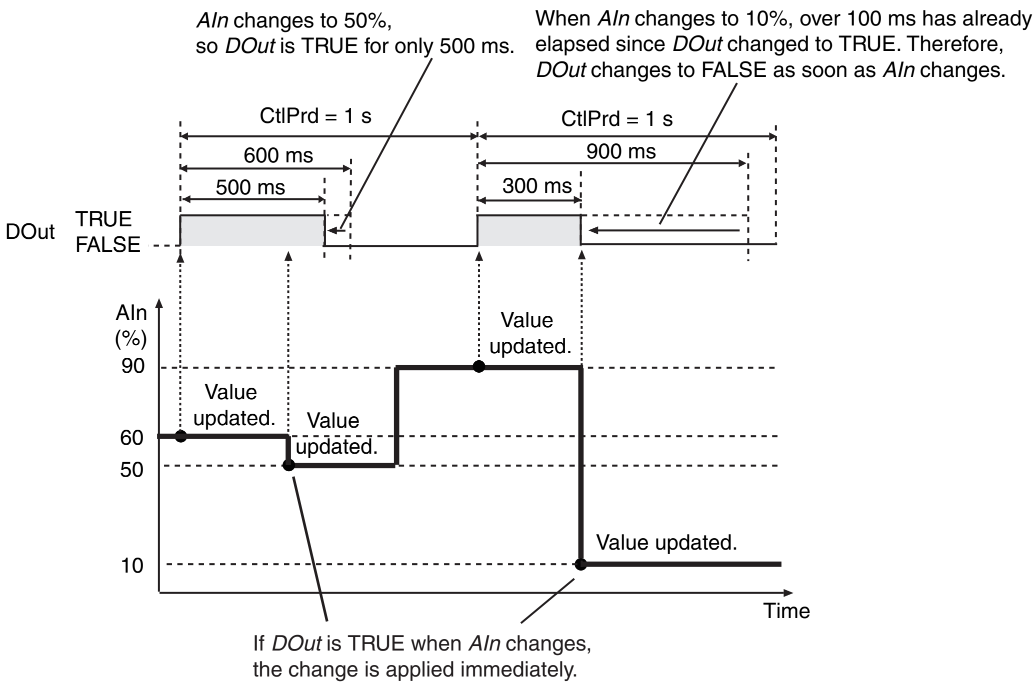


● *DOut* = TRUE

While *DOut* is TRUE, any change in the value of AIn is applied immediately.

For example, the following figure shows the operation when the value of control period *CtlPrd* is 1 s.

- If the value of *AIn* is 60% at the start of the control period and it changes to 50% while *DOut* is TRUE, *DOut* is TRUE for only 500 ms.
- Assume that the value of *AIn* was 90% at the start of the control period, that *DOut* changes to TRUE, and that 300 ms later *AIn* changes to 10%. In this case, 100 ms, which is 10%, has already elapsed, so *DOut* changes to FALSE immediately.



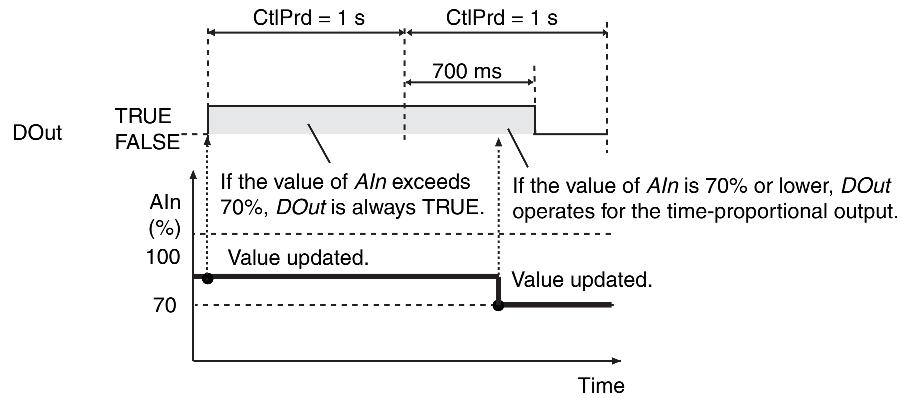
Operation of Time-proportional Output *DOut* for Minimum Pulse Width *MinPlsWidth*

The minimum pulse width is the minimum time that *DOut* will retain a value of TRUE or FALSE. You can set minimum pulse width *MinPlsWidth* to reduce chattering in *DOut*. For example, if the number of times a fan is turned ON and OFF is reduced in cooling control, power consumption is reduced.

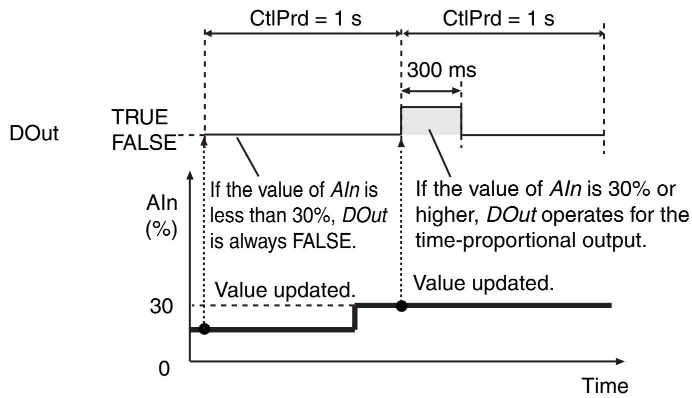
The following table shows the operation of *DOut* for the relationship between the values of *MinPlsWidth* and *AIn*.

Relationship between the values of ***MinPlsWidth*** and ***AIn***	Operation of ***DOut***
AIn < MinPlsWidth	Always FALSE
MinPlsWidth ≤ AIn ≤ 100 – MinPlsWidth	Time-proportional output
AIn > 100 – MinPlsWidth	Always TRUE

For example, the following figure shows the operation of *DOut* when *MinPlsWidth* is 30%. If the value of *AIn* is greater than 70%, *DOut* is always TRUE. When *AIn* decreases to 70% or lower, *DOut* operates for the time-proportional output.



If the value of *AIn* is less than 30%, *DOut* is always FALSE. When *AIn* increases to 30% or higher, DOut operates for the time-proportional output.



Operation of Time-proportional Output *DOut* for *Delay*

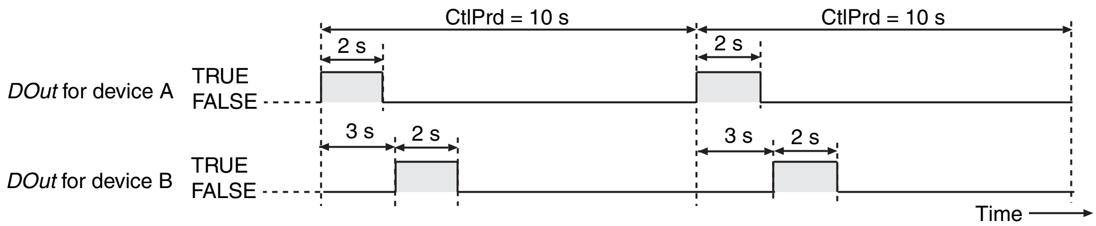
The delay prevents *DOut* from changing to TRUE until the set time has elapsed from the start of the control period. If more than one of this instruction is used, you can offset the timing of when *DOut* changes to TRUE by setting *Delay*. This reduces the chance that *DOut* will turn ON simultaneously for more than one instruction. For example, if you operate more than one heating device, you can use *Delay* to offset when the output to each heating device turns ON to reduce the power that is used at any one time.

DOut changes to TRUE after the percentage of time specified with *Delay* elapses from the start of the control period.

For example, you could set the following values for devices A and B, which have the same control period.

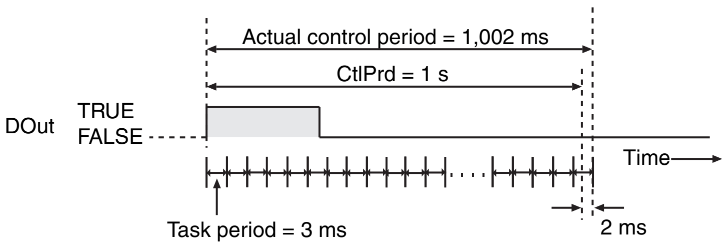
Device	Value of *Delay*	Value of *AIn*	Value of *CtlPrd*
Device A	0%	20%	10 s
Device B	30%		

DOut for device A changes to TRUE at the start of the control period. *DOut* for device B changes to TRUE three seconds after the start of the control period.



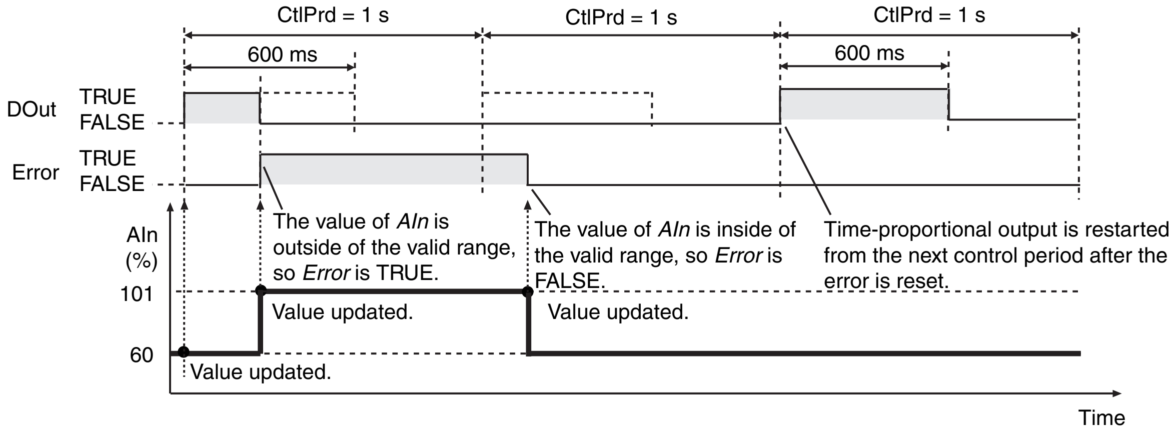
Precautions for Correct Use

- Set the value of control period *CtlPrd* to a multiple of the task period of the task to which the program is assigned. If the task period is not set to a multiple of *CtlPrd*, the actual control period will be from when control period *CtlPrd* ends until the next time the task is executed. For example, if the task period is set to 3 ms and the value of *CtlPrd* is 1 s, the actual control period will be 1,002 ms (from when *CtlPrd* ends until the next time the task is executed).

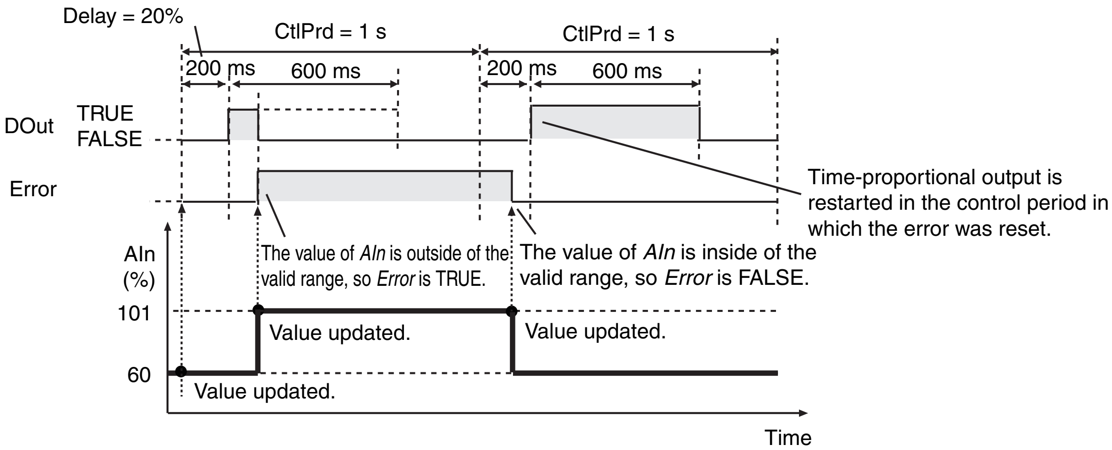


- Set the task period and control period *CtlPrd* so that the resolution of DOut is 0.1% or less. If the resolution of DOut exceeds 0.1%, the error between the ratio when *DOut* is TRUE and the value of *AIn* will be excessive and control performance will decrease. For example, if *CtlPrd* is 10 s, set the task period to 10 ms or lower.
- If you use more than one of this instruction and need to synchronize the control periods, use the instructions in the same program. If you use them in different programs, the control periods will depend on the timing of the execution of the programs, and they will not be synchronized.
- The time from when the value of *Enable* changes to TRUE and operation starts for *DOut* is not constant.

- An error occurs if the value of *AIn*, *CtlPrd*, *MinPlsWidth*, or *Delay* is outside of the valid range. *Error* changes to TRUE and *DOut* changes to FALSE. If the value of *AIn* exceeds the valid range, the operation of *DOut* will be as shown below depending on when the error is reset.
- If the error is reset after the point where *DOut* changes to TRUE, the time-proportional output for *DOut* is restarted from the next control period.



- If the error is reset before the point where *DOut* changes to TRUE, the time-proportional output for *DOut* is restarted in the control period in which the error was reset.

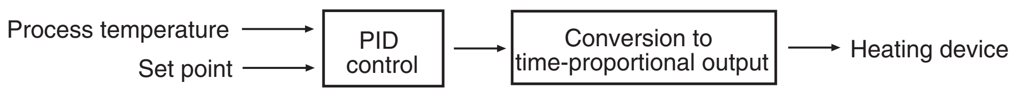


Version Information

A CPU Unit with unit version 1.02 or later and Sysmac Studio version 1.03 or higher are required to use this instruction.

Sample Programming

This sample performs temperature control for four points with upper/lower limit alarms and upper/lower deviation alarms. PID control is performed. The manipulated variables of PID control are converted to time-proportional output values that are output to heating devices.



Specifications

Temperature control is performed according to the following specifications.

Item	Specification
Input Unit	CJ1W-PH41U Isolated-type Universal Input Unit
Input types	K thermocouples
Output Unit	CJ1W-OD212 Transistor Output Unit
Set point	100°C
Upper limit of temperature	200°C
Lower limit of temperature	0°C
Hysteresis of upper/lower limit alarm	5°C
Upper deviation temperature	50°C
Lower deviation temperature	50°C
Hysteresis of upper/lower deviation alarm	3°C
Sampling period for PID control	100 ms
Output control period	1 s

Configuration and Settings

The following settings are used for the CJ1W-PH41U Input Unit.

Item	Set value
Input1:Input signal type	K(1)
Input2:Input signal type	K(1)
Input3:Input signal type	K(1)
Input4:Input signal type	K(1)

The following I/O map settings are used.

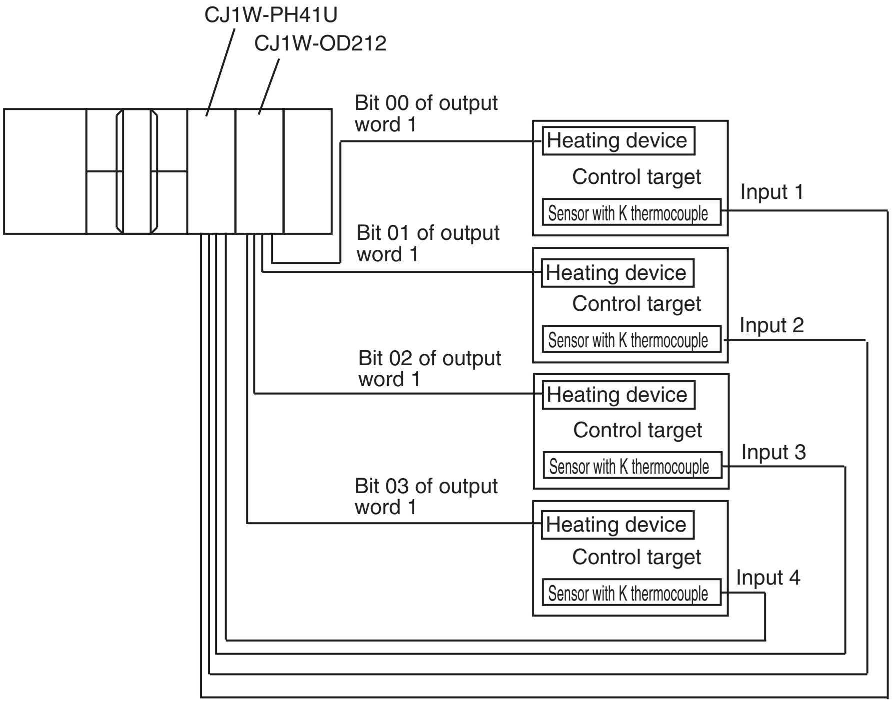
Unit	I/O port	Description	Variable
CJ1W-PH41U	Ch1_AIInPV	Process value for input 1 (INT data)	AI1
	Ch2_AIInPV	Process value for input 2 (INT data)	AI2
	Ch3_AIInPV	Process value for input 3 (INT data)	AI3
	Ch4_AIInPV	Process value for input 4 (INT data)	AI4
CJ1W-OD212	Ch1_Out00	Bit 00 of output word 1	DO1
	Ch1_Out01	Bit 01 of output word 1	DO2
	Ch1_Out02	Bit 02 of output word 1	DO3
	Ch1_Out03	Bit 03 of output word 1	DO4

The inputs and outputs for the temperature control for the four points correspond as shown below.

Input	Output
AI1	DO1
AI2	DO2
AI3	DO3
AI4	DO4

The task period of the task to which the program is assigned is 1 ms.

- **Configuration Diagram**

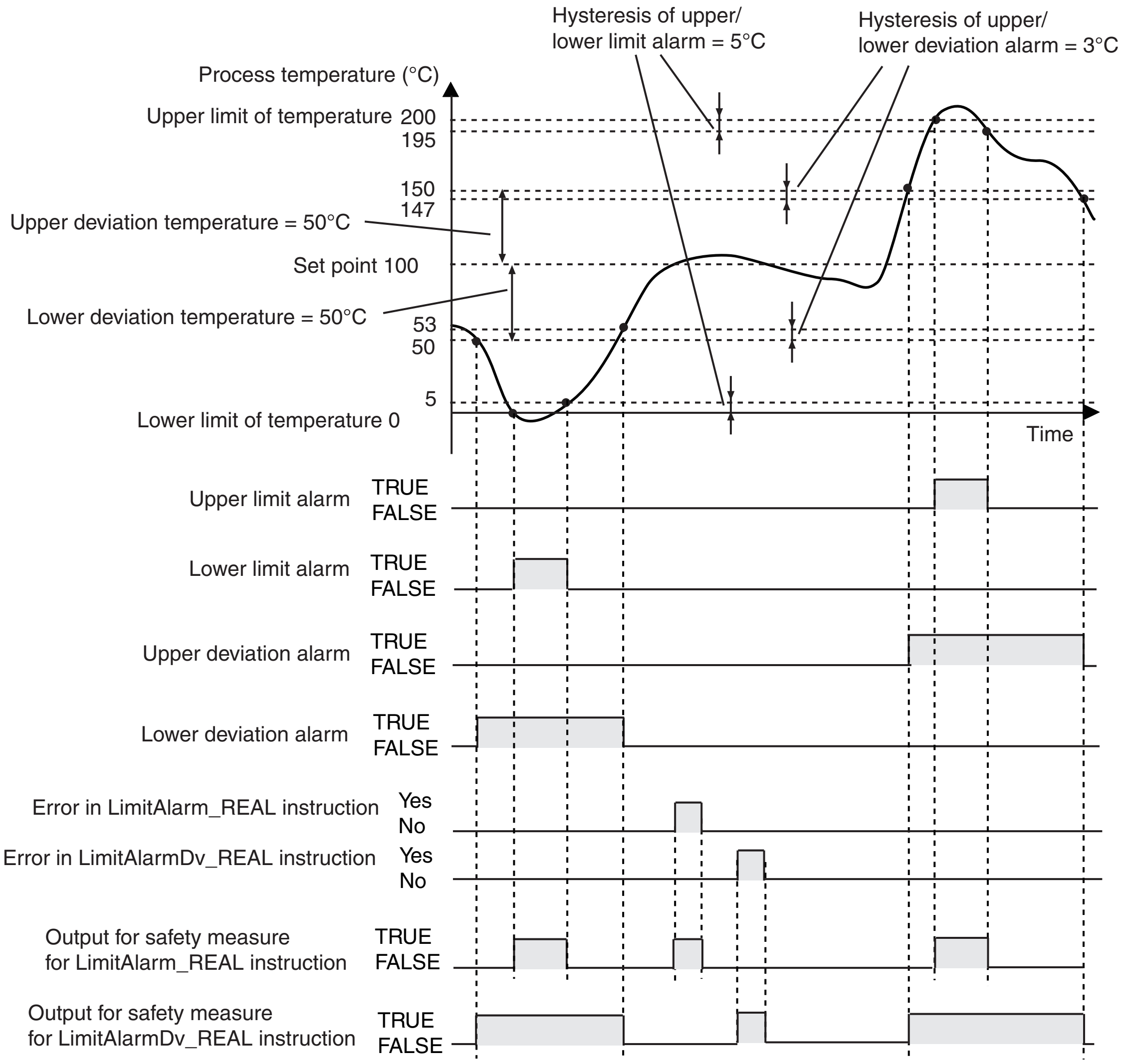


Processing

Perform the following procedure for all four points.

1. Get the process temperature.
2. Use the LimitAlarm_REAL instruction to output upper/lower limit alarms for the process temperature.
3. Perform an output as a safety measure if an error occurs in the LimitAlarm_REAL instruction or if an upper/lower limit alarm occurs.
4. Use the LimitAlarmDv_REAL instruction to output upper/lower deviation alarms for the deviation between the set point and the process temperature.
5. Perform an output as a safety measure if an error occurs in the LimitAlarmDv_REAL instruction or if an upper/lower deviation alarm occurs.
6. Perform temperature control with the PIDAT instruction.
7. Use the TimeProportionalOut instruction to output the manipulated variable as a time-proportional value to the heating device.

- **Operation of Upper/Lower Limit Alarms and Upper/Lower Deviation Alarms**



Application Programming

Definitions of Global Variables

Global Variables

Name	Data type	AT*1	Comment
AI1	INT	IOBus://rack#0/slot#0/Ch1_AIInPV	Process value for input 1 (INT data)
AI2	INT	IOBus://rack#0/slot#0/Ch2_AIInPV	Process value for input 2 (INT data)
AI3	INT	IOBus://rack#0/slot#0/Ch3_AIInPV	Process value for input 3 (INT data)
AI4	INT	IOBus://rack#0/slot#0/Ch4_AIInPV	Process value for input 4 (INT data)
DO1	BOOL	IOBus://rack#0/slot#1/Ch1_Out/Ch1_Out00	Bit 00 of output word 1
DO2	BOOL	IOBus://rack#0/slot#1/Ch1_Out/Ch1_Out01	Bit 01 of output word 1
DO3	BOOL	IOBus://rack#0/slot#1/Ch1_Out/Ch1_Out02	Bit 02 of output word 1
DO4	BOOL	IOBus://rack#0/slot#1/Ch1_Out/Ch1_Out03	Bit 03 of output word 1

*1. AT when the CJ1W-PH41U Unit is mounted to slot number 0 in rack number 0 and the CJ1W-OD212 Unit is mounted to slot number 1 in rack number 0.

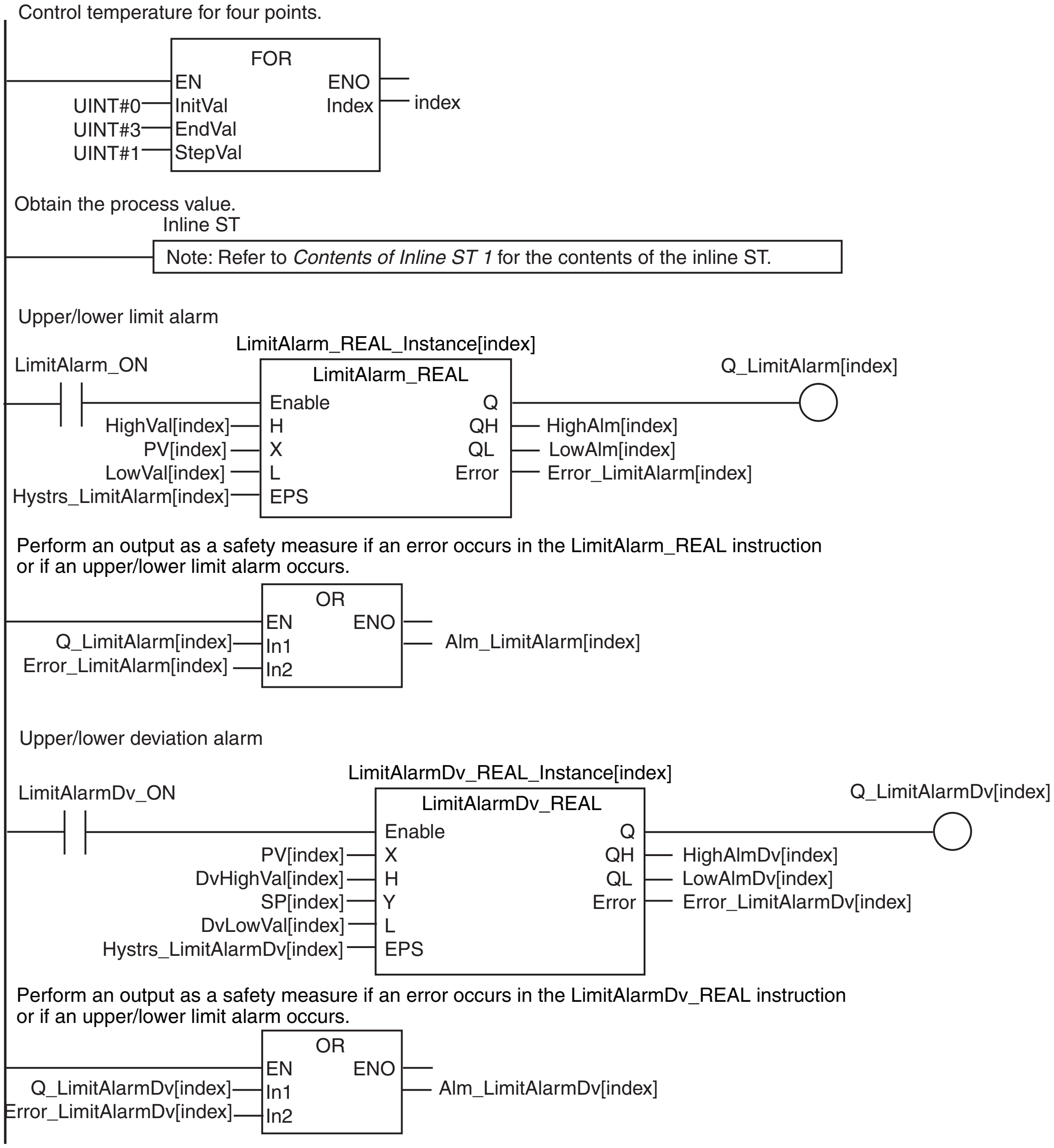
Note The global variables that are assigned to an I/O port of a Unit are automatically created according to the I/O map settings.

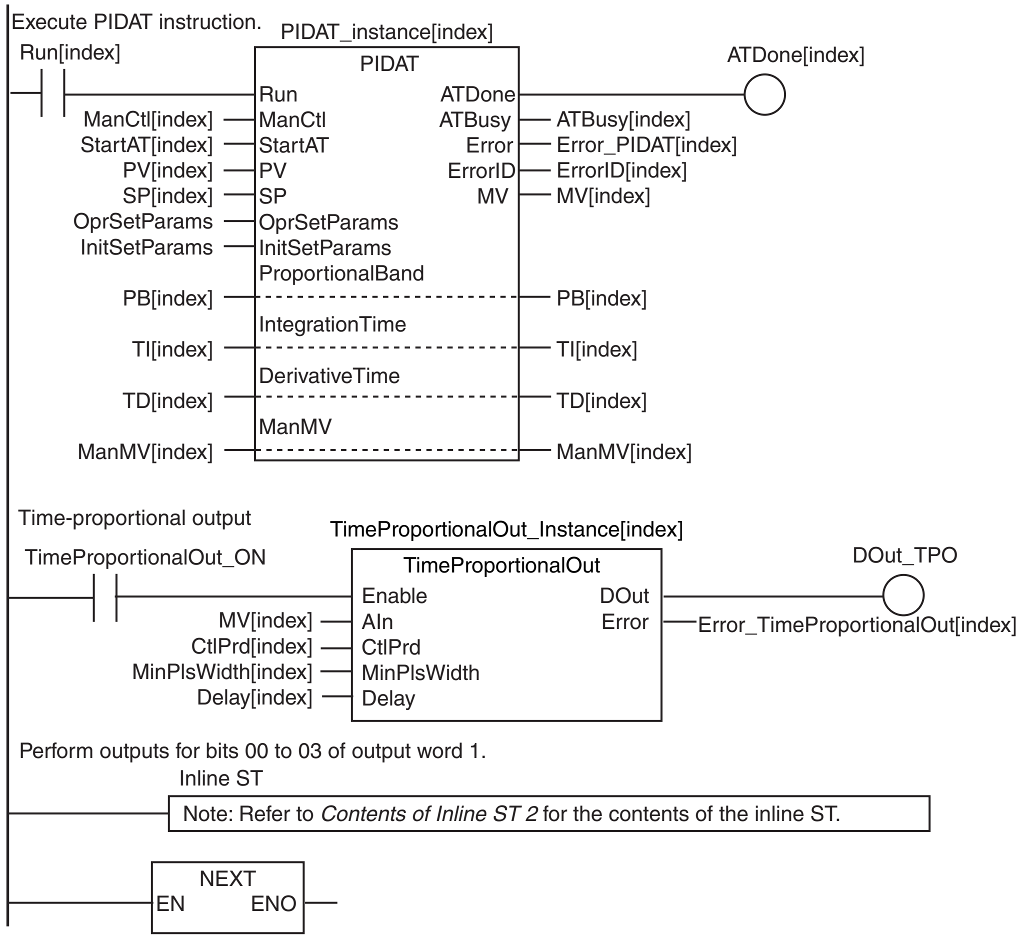
LD

Internal Variables	Name	Data type	Default	Retain	Comment
	index	UINT	0		Loop index
	LimitAlarm_ON	BOOL	True		Execution of Upper/Lower Limit Alarm instruction
	LimitAlarmDv_ON	BOOL	True		Execution of Upper/Lower Deviation Alarm instruction
	TimeProportionalOut_ON	BOOL	True		Execution of Time-proportional Output instruction
	AI	INT	0		Present value
	PV	ARRAY[0..3] OF REAL	[4(0.0)]		Process value
	SP	ARRAY[0..3] OF REAL	[4(100)]		Set point
	DOut_TPO	BOOL	False		Time-proportional output
	HighVal	ARRAY[0..3] OF REAL	[4(200)]		Upper limit set value of upper/lower limit alarm
	LowVal	ARRAY[0..3] OF REAL	[4(0.0)]		Lower limit set value of upper/lower limit alarm
	Hystrs_LimitAlarm	ARRAY[0..3] OF REAL	[4(5)]		Hysteresis of upper/lower limit alarm
	Q_LimitAlarm	ARRAY[0..3] OF BOOL	[4(False)]		Upper/lower limit alarm output
	HighAlm	ARRAY[0..3] OF BOOL	[4(False)]		Upper limit alarm
	LowAlm	ARRAY[0..3] OF BOOL	[4(False)]		Lower limit alarm
	Error_LimitAlarm	ARRAY[0..3] OF BOOL	[4(False)]		Error in LimitAlarm_REAL instruction
	Alm_LimitAlarm	ARRAY[0..3] OF BOOL	[4(False)]		Output for safety measure for Upper/Lower Limit Alarm instruction
	DvHighVal	ARRAY[0..3] OF REAL	[4(50)]		Upper deviation set value of upper/lower deviation alarm
	DvLowVal	ARRAY[0..3] OF REAL	[4(50)]		Lower deviation set value of upper/lower deviation alarm
	Q_LimitAlarmDv	ARRAY[0..3] OF BOOL	[4(False)]		Upper/lower deviation alarm output
	HighAlmDv	ARRAY[0..3] OF BOOL	[4(False)]		Upper deviation alarm
	LowAlmDv	ARRAY[0..3] OF BOOL	[4(False)]		Lower deviation alarm
	Error_LimitAlarmDv	ARRAY[0..3] OF BOOL	[4(False)]		Error in LimitAlarmDv_REAL instruction
	Hystrs_LimitAlarmDv	ARRAY[0..3] OF REAL	[4(3)]		Hysteresis of upper/lower deviation alarm
	Alm_LimitAlarmDv	ARRAY[0..3] OF BOOL	[4(False)]		Output for safety measure for Upper/Lower Deviation Alarm instruction

Internal Variables	Name	Data type	Default	Retain	Comment
	Run	ARRAY[0..3] OF BOOL	[4(False)]		Execution condition
	ManCtl	ARRAY[0..3] OF BOOL	[4(False)]		Manual/auto control
	StartAT	ARRAY[0..3] OF BOOL	[4(False)]		Autotuning execution condition
	OprSetParams	_sOPR_SET_PARAMS	(MVLowLmt:=0.0, MVUpLmt:=100.0, ManResetVal:=0.0, MVTrackSw:=False, MVTrackVal:=0.0, StopMV:=0.0, ErrorMV:=0.0, Alpha:=0.65, ATCalcGain:=1.0, ATHystrs:=0.2)		Operation setting parameters
	InitSetParams	_sINIT_SET_PARAMS	(SampTime:=T#100ms, RngLowLmt:=–10.0, RngUpLmt:=1000.0, DirOpr:=False)		Initial setting parameters
	PB	ARRAY[0..3] OF REAL	[4(10)]	✔	Proportional band
	TI	ARRAY[0..3] OF TIME	[4(T#0S)]	✔	Integration time
	TD	ARRAY[0..3] OF TIME	[4(T#0S)]	✔	Derivative time
	ManMV	ARRAY[0..3] OF REAL	[4(0.0)]		Manual manipulated variable
	ATDone	ARRAY[0..3] OF BOOL	[4(False)]		Autotuning normal completion
	ATBusy	ARRAY[0..3] OF BOOL	[4(False)]		Autotuning busy
	Error_PIDAT	ARRAY[0..3] OF BOOL	[4(False)]		Error in PIDAT instruction
	ErrorID	ARRAY[0..3] OF WORD	[4(16#0)]		Error ID for PIDAT instruction
	MV	ARRAY[0..3] OF REAL	[4(0.0)]		Manipulated variable
	CtlPrd	ARRAY[0..3] OF TIME	[4(T#1s)]		Control period
	MinPlsWidth	ARRAY[0..3] OF REAL	[4(0.0)]		Minimum pulse width
	Delay	ARRAY[0..3] OF REAL	[4(0.0)]		ON-delay time
	Error_TimeProportionalOut	ARRAY[0..3] OF BOOL	[4(False)]		Error in TimeProportionalOut instruction
	LimitAlarm_REAL_instance	ARRAY[0..3] OF LimitAlarm_REAL			
	LimitAlarmDv_REAL_instance	ARRAY[0..3] OF LimitAlarmDv_REAL			
	PIDAT_instance	ARRAY[0..3] OF PIDAT			
	TimeProportionalOut_instance	ARRAY[0..3] OF TimeProportionalOut			

External Variables	Name	Data type	Comment
	AI1	INT	Process value for input 1
	AI2	INT	Process value for input 2
	AI3	INT	Process value for input 3
	AI4	INT	Process value for input 4
	DO1	BOOL	Bit 00 of output word 1
	DO2	BOOL	Bit 01 of output word 1
	DO3	BOOL	Bit 02 of output word 1
	DO4	BOOL	Bit 03 of output word 1


Control temperature for four points.
FOR
EN
ENO
UINT#0
InitVal
Index
index
UINT#3
EndVal
UINT#1
StepVal
Obtain the process value.
Inline ST
Note: Refer to Contents of Inline ST 1 for the contents of the inline ST.
Upper/lower limit alarm
LimitAlarm_REAL_Instance[index]
LimitAlarm_ON
LimitAlarm_REAL
Q_LimitAlarm[index]
Enable
Q
HighVal[index]
H
QH
HighAlm[index]
PV[index]
X
QL
LowAlm[index]
LowVal[index]
L
Error
Error_LimitAlarm[index]
Hystrs_LimitAlarm[index]
EPS
Perform an output as a safety measure if an error occurs in the LimitAlarm_REAL instruction or if an upper/lower limit alarm occurs.
OR
EN
ENO
Q_LimitAlarm[index]
In1
Alm_LimitAlarm[index]
Error_LimitAlarm[index]
In2
Upper/lower deviation alarm
LimitAlarmDv_REAL_Instance[index]
LimitAlarmDv_ON
LimitAlarmDv_REAL
Q_LimitAlarmDv[index]
Enable
Q
PV[index]
X
QH
HighAlmDv[index]
DvHighVal[index]
H
QL
LowAlmDv[index]
SP[index]
Y
Error
Error_LimitAlarmDv[index]
DvLowVal[index]
L
Hystrs_LimitAlarmDv[index]
EPS
Perform an output as a safety measure if an error occurs in the LimitAlarmDv_REAL instruction or if an upper/lower limit alarm occurs.
OR
EN
ENO
Q_LimitAlarmDv[index]
In1
Alm_LimitAlarmDv[index]
Error_LimitAlarmDv[index]
In2



● Contents of Inline ST 1

```
// Get values of inputs 1 to 4.
CASE index OF
  INT#0:
    AI:=AI1;
  INT#1:
    AI:=AI2;
  INT#2:
    AI:=AI3;
  ELSE
    AI:=AI4;
END_CASE;

// Convert PV AI to real number.
PV[index]:=INT_TO_REAL(AI)/REAL#10.0;
// CJ1W-PH41U output is ten times the process value, so divide by 10.0.
```

● Contents of Inline ST 2

```
// Perform outputs for bits 00 to 03 of output word 1.
CASE index OF
  INT#0:
    DO1:=DOut_TPO;
  INT#1:
    DO2:=DOut_TPO;
  INT#2:
    DO3:=DOut_TPO;
  ELSE
    DO4:=DOut_TPO;
END_CASE;
```

ST

Internal Variables	Name	Data type	Default	Retain	Comment
	index	UINT	0		Loop index
	LimitAlarm_ON	BOOL	True		Execution of Upper/Lower Limit Alarm instruction
	LimitAlarmDv_ON	BOOL	True		Execution of Upper/Lower Deviation Alarm instruction
	TimeProportionalOut_ON	BOOL	True		Execution of Time-proportional Output instruction
	AI	INT	0		Present value
	PV	ARRAY[0..3] OF REAL	[4(0.0)]		Process value
	SP	ARRAY[0..3] OF REAL	[4(100)]		Set point
	DOut_TPO	BOOL	False		Time-proportional output
	HighVal	ARRAY[0..3] OF REAL	[4(200)]		Upper limit set value of upper/lower limit alarm
	LowVal	ARRAY[0..3] OF REAL	[4(0.0)]		Lower limit set value of upper/lower limit alarm
	Hystrs_LimitAlarm	ARRAY[0..3] OF REAL	[4(5)]		Hysteresis of upper/lower limit alarm
	Q_LimitAlarm	ARRAY[0..3] OF BOOL	[4(False)]		Upper/lower limit alarm output
	HighAlm	ARRAY[0..3] OF BOOL	[4(False)]		Upper limit alarm
	LowAlm	ARRAY[0..3] OF BOOL	[4(False)]		Lower limit alarm
	Error_LimitAlarm	ARRAY[0..3] OF BOOL	[4(False)]		Error in LimitAlarm_REAL instruction
	Alm_LimitAlarm	ARRAY[0..3] OF BOOL	[4(False)]		Output for safety measure for Upper/Lower Limit Alarm instruction
	DvHighVal	ARRAY[0..3] OF REAL	[4(50)]		Upper deviation set value of upper/lower deviation alarm
	DvLowVal	ARRAY[0..3] OF REAL	[4(50)]		Lower deviation set value of upper/lower deviation alarm
	Q_LimitAlarmDv	ARRAY[0..3] OF BOOL	[4(False)]		Upper/lower deviation alarm output
	HighAlmDv	ARRAY[0..3] OF BOOL	[4(False)]		Upper deviation alarm
	LowAlmDv	ARRAY[0..3] OF BOOL	[4(False)]		Lower deviation alarm
	Error_LimitAlarmDv	ARRAY[0..3] OF BOOL	[4(False)]		Error in LimitAlarmDv_REAL instruction
	Hystrs_LimitAlarmDv	ARRAY[0..3] OF REAL	[4(3)]		Hysteresis of upper/lower deviation alarm
	Alm_LimitAlarmDv	ARRAY[0..3] OF BOOL	[4(False)]		Output for safety measure for Upper/Lower Deviation Alarm instruction
	Run	ARRAY[0..3] OF BOOL	[4(False)]		Execution condition
	ManCtl	ARRAY[0..3] OF BOOL	[4(False)]		Manual/auto control
	StartAT	ARRAY[0..3] OF BOOL	[4(False)]		Autotuning execution condition

Internal Variables	Name	Data type	Default	Retain	Comment
	OprSetParams	_sOPR_SET_PARAMS	(MVLowLmt:=0.0, MVUpLmt:=100.0, ManReset-Val:=0.0, MVTrackSw:=False, MVTrackVal:=0.0, StopMV:=0.0, ErrorMV:=0.0, Alpha:=0.65, ATCalcGain:=1.0, ATHystrs:=0.2)		Operation setting parameters
	InitSetParams	_sINIT_SET_PARAMS	(SampTime:=T#100ms, RngLowLmt:=–10.0, RngUpLmt:=1000.0, DirOpr:=False)		Initial setting parameters
	PB	ARRAY[0..3] OF REAL	[4(10)]	✔	Proportional band
	TI	ARRAY[0..3] OF TIME	[4(T#0S)]	✔	Integration time
	TD	ARRAY[0..3] OF TIME	[4(T#0S)]	✔	Derivative time
	ManMV	ARRAY[0..3] OF REAL	[4(0.0)]		Manual manipulated variable
	ATDone	ARRAY[0..3] OF BOOL	[4(False)]		Autotuning normal completion
	ATBusy	ARRAY[0..3] OF BOOL	[4(False)]		Autotuning busy
	Error_PIDAT	ARRAY[0..3] OF BOOL	[4(False)]		Error in PIDAT instruction
	ErrorID	ARRAY[0..3] OF WORD	[4(16#0)]		Error ID for PIDAT instruction
	MV	ARRAY[0..3] OF REAL	[4(0.0)]		Manipulated variable
	CtlPrd	ARRAY[0..3] OF TIME	[4(T#1s)]		Control period
	MinPlsWidth	ARRAY[0..3] OF REAL	[4(0.0)]		Minimum pulse width
	Delay	ARRAY[0..3] OF REAL	[4(0.0)]		ON-delay time
	Error_TimeProportionalOut	ARRAY[0..3] OF BOOL	[4(False)]		Error in TimeProportionalOut instruction
	LimitAlarm_REAL_instance	ARRAY[0..3] OF LimitAlarm_REAL			
	LimitAlarmDv_REAL_instance	ARRAY[0..3] OF LimitAlarmDv_REAL			
	PIDAT_instance	ARRAY[0..3] OF PIDAT			
	TimeProportionalOut_instance	ARRAY[0..3] OF TimeProportionalOut			

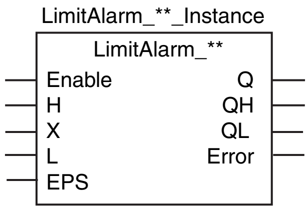
External Variables	Name	Data type	Comment
	AI1	INT	Process value for input 1
	AI2	INT	Process value for input 2
	AI3	INT	Process value for input 3
	AI4	INT	Process value for input 4
	DO1	BOOL	Bit 00 of output word 1
	DO2	BOOL	Bit 01 of output word 1
	DO3	BOOL	Bit 02 of output word 1
	DO4	BOOL	Bit 03 of output word 1

```
// Control temperature for four points.
FOR index:=UINT#0 TO UINT#3 BY UINT#1 DO

  // Get values of inputs 1 to 4.
     CASE index OF
      INT#0:
        AI:=AI1;
      INT#1:
        AI:=AI2;
      INT#2:
        AI:=AI3;
      ELSE
        AI:=AI4;
    END_CASE;

  // Convert PV AI to real number.
  PV[index]:=INT_TO_REAL(AI)/REAL#10.0; // CJ1W-PH41U output is ten times
  // the process value, so divide by 10.0.

  // Upper/lower limit alarm
  LimitAlarm_REAL_instance[index](
    Enable :=LimitAlarm_ON,
    H      :=HighVal[index],
    X      :=PV[index],
    L      :=LowVal[index],
    EPS    :=Hystrs_LimitAlarm[index],
    Q      =>Q_LimitAlarm[index],
    QH     =>HighAlm[index],
    QL     =>LowAlm[index],
    Error  =>Error_LimitAlarm[index]);

 // Perform an output as a safety measure if an error occurs in the
 // LimitAlarmDv_REAL instruction or if an upper/lower limit alarm occurs.
  Alm_LimitAlarm[index]:=Q_LimitAlarm[index] OR Error_LimitAlarm[index];

  // Upper/lower deviation alarm
  LimitAlarmDv_REAL_instance[index](
    Enable :=LimitAlarmDv_ON,
    X      :=PV[index],
    H      :=DvHighVal[index],
    Y      :=SP[index],
    L      :=DvLowVal[index],
    EPS    :=Hystrs_LimitAlarmDv[index],
    Q       =>Q_LimitAlarmDv[index],
    QH      =>HighAlmDv[index],
    QL      =>LowAlmDv[index],
    Error   =>Error_LimitAlarmDv[index]);

  // Perform an output as a safety measure if an error occurs in the
  // LimitAlarmDv_REAL instruction or if an upper/lower limit alarm occurs.
  Alm_LimitAlarmDv[index]:=Q_LimitAlarmDv[index] OR Error_LimitAlarmDv[index];

  // Execute PIDAT instruction.
  PIDAT_instance[index](
    Run              :=Run[index],
    ManCtl           :=ManCtl[index],
    StartAT          :=StartAT[index],
    PV               :=PV[index],
    SP               :=SP[index],
    OprSetParams     :=OprSetParams,
    InitSetParams    :=InitSetParams,
    ProportionalBand:=PB[index],
    IntegrationTime :=TI[index],
    DerivativeTime  :=TD[index],
    ManMV            :=ManMV[index],
```

```
    ATDone           =>ATDone[index],
    ATBusy           =>ATBusy[index],
    Error            =>Error_PIDAT[index],
    ErrorID          =>ErrorID[index],
    MV               =>MV[index]);

  // Time-proportional output
  TimeProportionalOut_instance[index](
    Enable      :=TimeProportionalOut_ON,
    AIn         :=MV[index],
    CtlPrd      :=CtlPrd[index],
    MinPlsWidth :=MinPlsWidth[index],
    Delay       :=Delay[index],
    DOut         =>DOut_TPO,
    Error        =>Error_TimeProportionalOut[index]);

  // Perform outputs for bits 00 to 03 of output word 1.
  CASE index OF
    INT#0:
      DO1:=DOut_TPO;
    INT#1:
      DO2:=DOut_TPO;
    INT#2:
      DO3:=DOut_TPO;
    ELSE
      DO4:=DOut_TPO;
  END_CASE;

END_FOR;
```

LimitAlarm_**

The LimitAlarm_** instruction outputs an alarm if the input value is below the lower limit set value or above the upper limit set value.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
LimitAlarm_**	Upper/Lower Limit Alarm Group	FB	 “**” must be REAL or LREAL.	LimitAlarm_**_instance(Enable, H, X, L, EPS, Q, QH, QL, Error); “**” must be REAL or LREAL.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Enable	Enable	Input	TRUE: Execute FALSE: Reset alarm	Depends on data type.	---	FALSE
H	Upper limit set value		Upper limit set value for the input value			0
X	Input value		Value to monitor			
L	Lower limit set value		Lower limit set value for the input value			
EPS	Hysteresis		Hysteresis of the alarm	Depends on data type.*		
Q	Alarm output	Output	TRUE: There is either an upper limit alarm or a lower limit alarm. FALSE: There is neither an upper limit alarm nor a lower limit alarm.	Depends on data type.	---	---
QH	Upper limit alarm		TRUE: There is an upper limit alarm. FALSE: There is no upper limit alarm.			
QL	Lower limit alarm		TRUE: There is a lower limit alarm. FALSE: There is no lower limit alarm.			

* Negative numbers are excluded.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Enable	OK																			
H														OK	OK					
X	Must be same data type as *H*.																			
L	Must be same data type as *H*.																			
EPS	Must be same data type as *H*.																			
Q	OK																			
QH	OK																			
QL	OK																			

Function

The LimitAlarm_** instruction monitors the input value to see if it is between the lower limit set value and the upper limit set value. The LimitAlarm_** instruction outputs an alarm if the input value is below the lower limit set value or above the upper limit set value. Use this instruction in temperature control, e.g., to monitor the process temperature.

Input value *X* is monitored while *Enable* is TRUE. If the value of *X* exceeds the value of upper limit set value *H*, upper limit alarm *QH* changes to TRUE. If the value of *X* goes below the value of lower limit set value *L*, lower limit alarm *QL* changes to TRUE. If the value of either *QH* or *QL* is TRUE, the value of alarm output *Q* is TRUE. The values of *X*, *H*, *L*, and hysteresis *EPS* are continuously updated while *Enable* is TRUE.

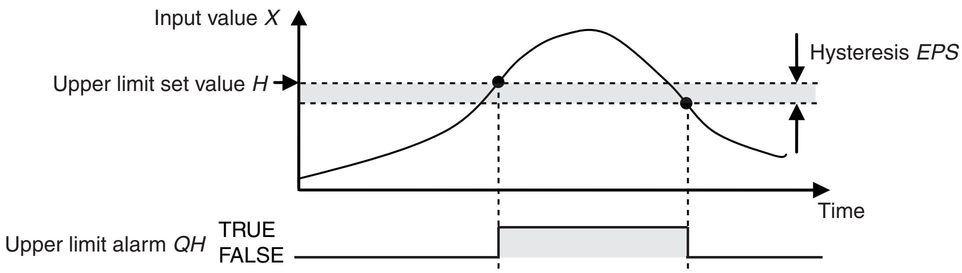
If *Enable* changes to FALSE, the alarm is reset. When the alarm is reset, *Q*, *QH*, and *QL* change to FALSE.

The data types of *H*, *X*, *L*, and *EPS* must be either REAL or LREAL. The name of the instruction is determined by the data types of *H*, *X*, *L*, and *EPS*. If the name of the instruction is LimitAlarm_LREAL, the data types of *H*, *X*, *L*, and *EPS* are all LREAL.

Operation of Upper Limit Alarm *QH*

The value of upper limit alarm *QH* changes as shown below. You can set the hysteresis to prevent hunting in the limit alarm.

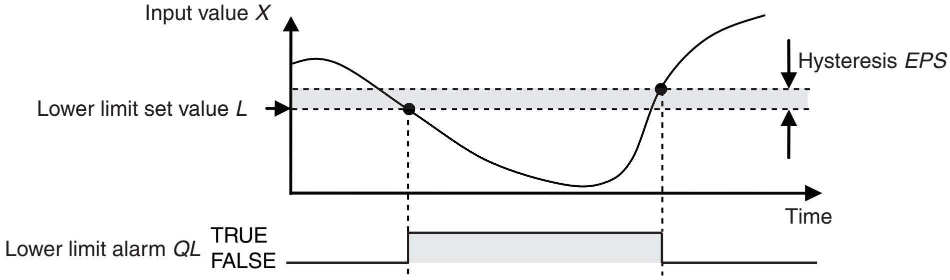
- If Input value *X* > Upper limit set value *H*, then *QH* is TRUE.
- If Input value *X* < Upper limit set value *H* – Hysteresis *EPS*, then *QH* is FALSE.



Operation of Lower Limit Alarm *QL*

The value of lower limit alarm *QL* changes as shown below. You can set the hysteresis to prevent hunting in the limit alarm.

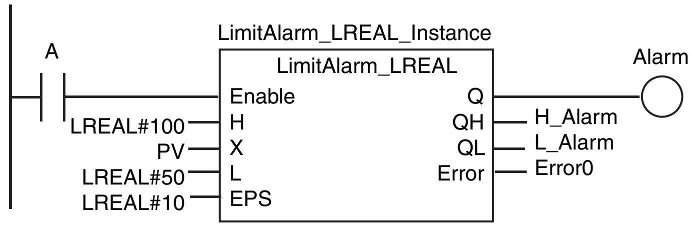
- If Input value *X* < Lower limit set value *L*, then *QL* is TRUE.
- If Input value *X* > Lower limit set value *L* + Hysteresis *EPS*, then *QL* is FALSE.



Notation Example

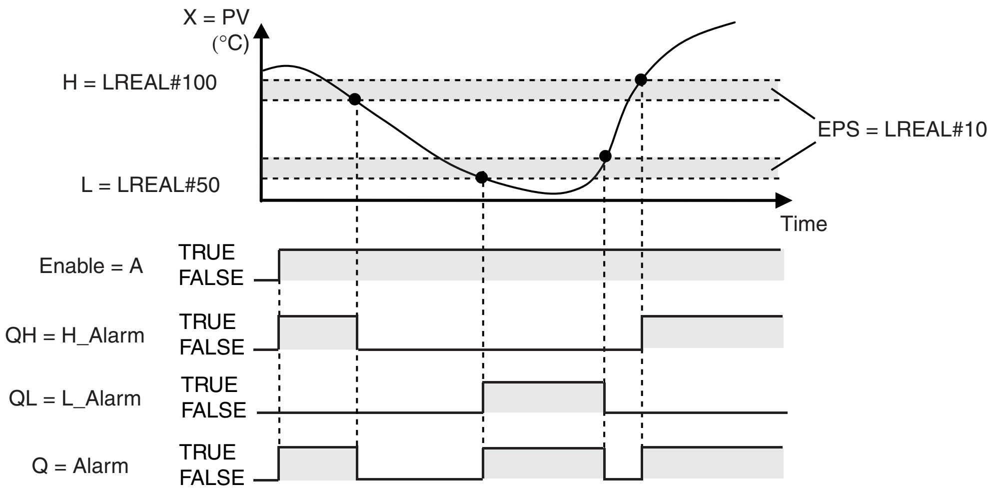
The following notation example sets upper limit set value *H* to 100°C, lower limit set value *L* to 50°C, and hysteresis *EPS* to 10°C.

LD



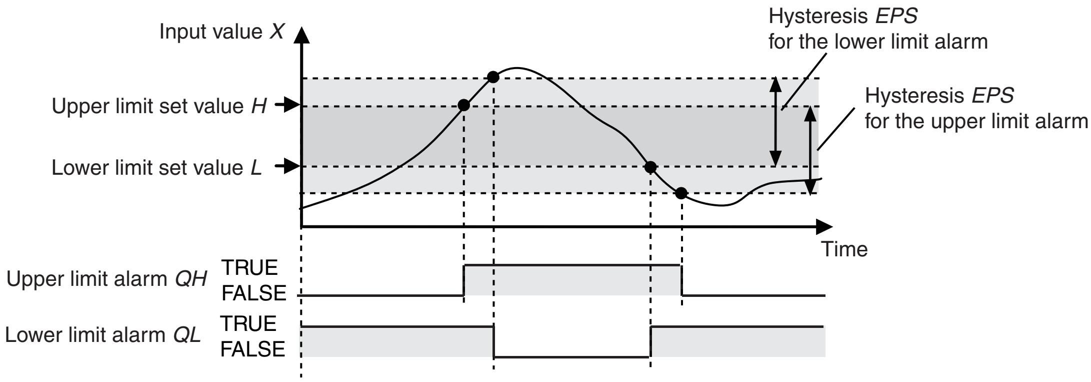
ST

```
LimitAlarm_LREAL_instance(A,LREAL#100,PV,LREAL#50,LREAL#10,Alarm,H_Alarm,L_Alarm,Error0);
```

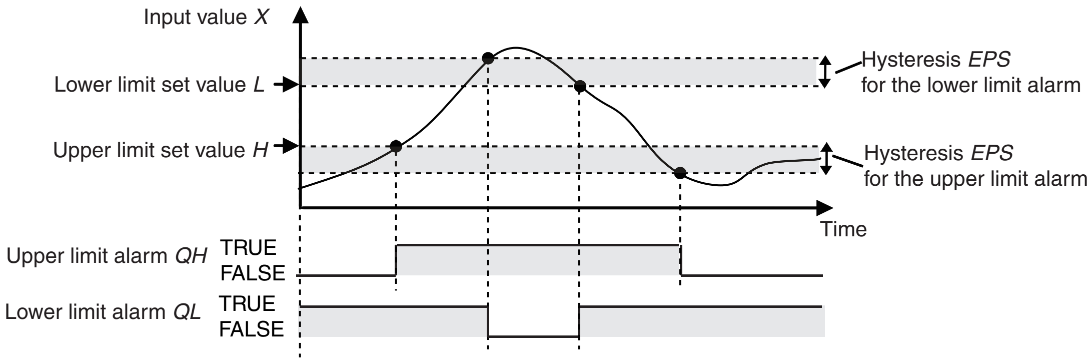


Additional Information

- Use the LimitAlarm_REAL instruction to reduce the instruction execution time.
- You can set *EPS* to less than *H* – *L*. If you do so, both *QH* and *QL* can be TRUE at the same time.



- You can set H and L so that *H* < *L*. If you do so, either *QH* or *QL* will always be TRUE.



Precautions for Correct Use

- An error occurs if the value of *EPS* is outside of the valid range. *Error* changes to TRUE, and *Q*, *QH*, and *QL* change to FALSE.
- You can use this instruction for safety measures, for example, to turn OFF a temperature control output when an alarm is output. If you do so, design the safety measures so that safety is maintained even when an error causes *Q*, *QH*, and *HL* to change to FALSE. For an application example, refer to the sample programming that is provided for the TimeProportionalOut instruction (page 2-733).

Version Information

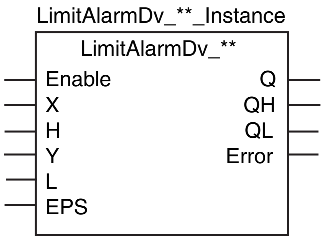
A CPU Unit with unit version 1.02 or later and Sysmac Studio version 1.03 or higher are required to use this instruction.

Sample Programming

Refer to the sample programming that is provided for the TimeProportionalOut instruction (page 2-733).

LimitAlarmDv_**

The LimitAlarmDv_** instruction outputs an alarm if the deviation in the input value from the reference value exceeds the lower deviation set value or the upper deviation set value.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
LimitAlarmDv_**	Upper/Lower Deviation Alarm Group	FB	 "**" must be REAL or LREAL.	LimitAlarmDv_**instance(Enable, X, H, Y, L, EPS, Q, QH, QL, Error); "**" must be REAL or LREAL.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Enable	Enable	Input	TRUE: Execute FALSE: Reset alarm	Depends on data type.	---	FALSE
X	Input value		Value to monitor			0
H	Upper deviation set value		Set value for an alarm for an upward deviation in respect to the reference value			
Y	Reference value		Reference value for deviation			
L	Lower deviation set value		Set value for an alarm for a downward deviation in respect to the reference value			
EPS	Hysteresis		Hysteresis of the alarm	Depends on data type.*		
Q	Deviation alarm output	Output	TRUE: There is either an upper deviation alarm or a lower deviation alarm. FALSE: There is neither an upper deviation alarm nor a lower deviation alarm.	Depends on data type.	---	---
QH	Upper deviation alarm		TRUE: There is an upper deviation alarm. FALSE: There is no upper deviation alarm.			
QL	Lower deviation alarm		TRUE: There is a lower deviation alarm. FALSE: There is no lower deviation alarm.			

* Negative numbers are excluded.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Enable	OK																			
X														OK	OK					
H	Must be same data type as *X*.																			
Y	Must be same data type as *X*.																			
L	Must be same data type as *X*.																			
EPS	Must be same data type as *X*.																			
Q	OK																			
QH	OK																			
QL	OK																			

Function

The LimitAlarmDv_** instruction monitors the deviation in the input value from the reference value to see if it exceeds the lower deviation set value or the upper deviation set value. If the deviation exceeds the lower deviation set value or the upper deviation set value, the instruction outputs an alarm. Use this instruction in temperature control, e.g., to monitor the deviation in the process temperature from the set point.

The deviation in input value *X* from the reference value *Y* is monitored while *Enable* is TRUE. If the upward deviation in *X* from *Y* exceeds the value of upper deviation set value *H*, upper deviation alarm *QH* changes to TRUE. If the downward deviation in *X* from *Y* exceeds the value of lower deviation set value *L*, lower deviation alarm *QL* changes to TRUE. If the value of either *QH* or *QL* is TRUE, the value of alarm output *Q* is TRUE. The values of *X*, *H*, *Y*, *L*, and hysteresis *EPS* are continuously updated while *Enable* is TRUE.

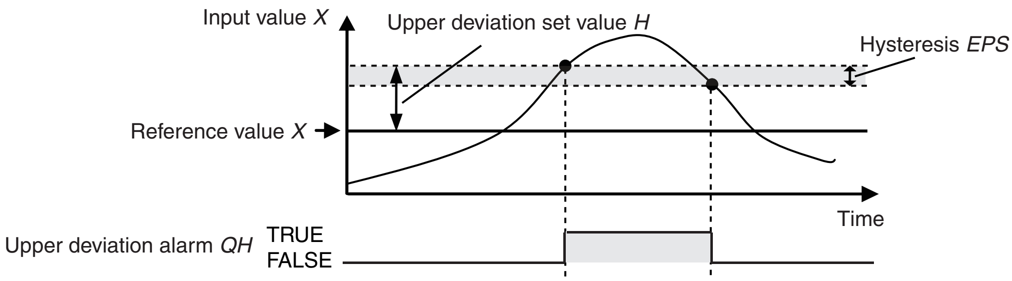
If *Enable* changes to FALSE, the alarm is reset. When the alarm is reset, *Q*, *QH*, and *QL* change to FALSE.

The data types of *X*, *H*, *Y*, *L*, and *EPS* must be either REAL or LREAL. The name of the instruction is determined by the data types of *X*, *H*, *Y*, *L*, and *EPS*. If the name of the instruction is LimitAlarmDv_LREAL, the data types of *X*, *H*, *Y*, *L*, and *EPS* are all LREAL.

Operation of Upper Deviation Alarm *QH*

Upper deviation alarm *QH* is the alarm for an upward deviation in respect to reference value *Y*. The value of *QH* changes as shown below. You can set the hysteresis to prevent hunting in the deviation alarm.

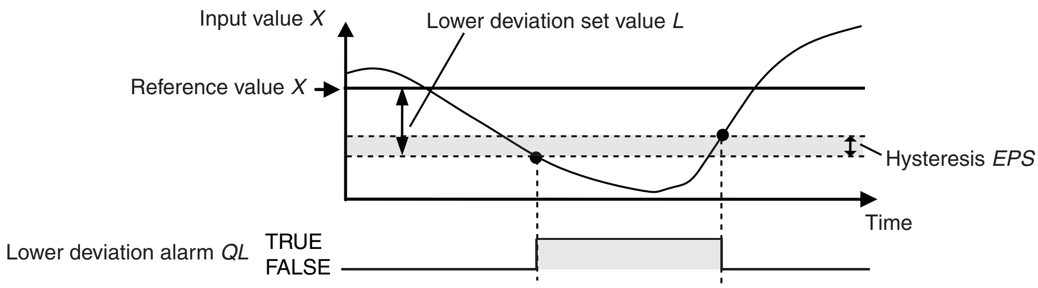
- If Input value *X* – Reference value *Y* > Upper deviation set value *H*, then *QH* is TRUE.
- If Input value *X* – Reference value *Y* < Upper deviation set value *H* – Hysteresis *EPS*, then *QH* is FALSE.



Operation of Lower Deviation Alarm *QL*

Lower deviation alarm *QL* is the alarm for a downward deviation in respect to reference value *Y*. The value of *QL* changes as shown below. You can set the hysteresis to prevent hunting in the deviation alarm.

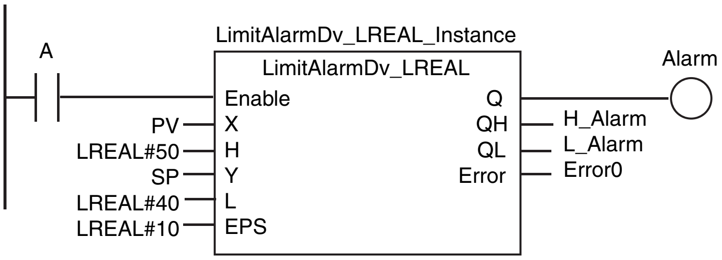
- If –(Input value *X* – Reference value *Y*) > Lower deviation set value *L*, then *QL* is TRUE.
- If –(Input value *X* – Reference value *Y*) < Lower deviation set value *L* – Hysteresis *EPS*, then *QL* is FALSE.



Notation Example

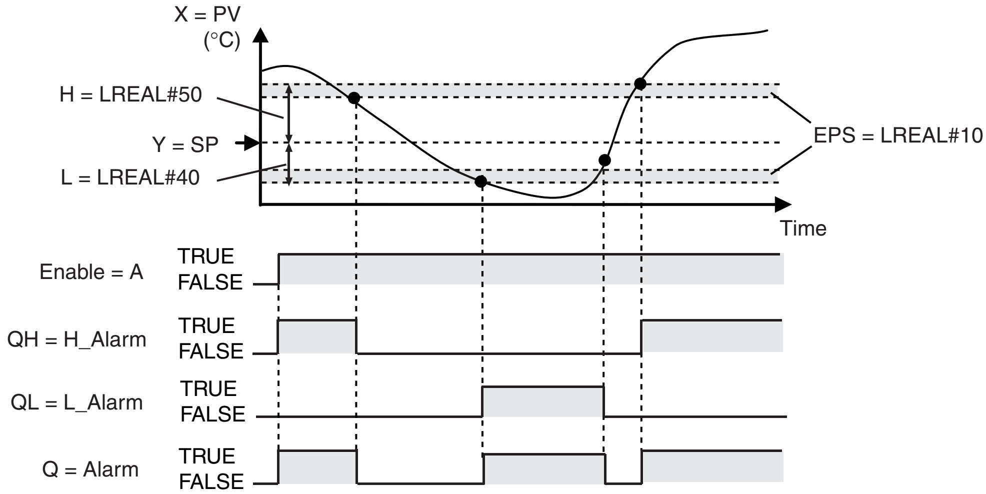
The following notation example sets upper deviation set value *H* to 50°C, lower deviation set value *L* to 40°C, and hysteresis *EPS* to 10°C.

LD



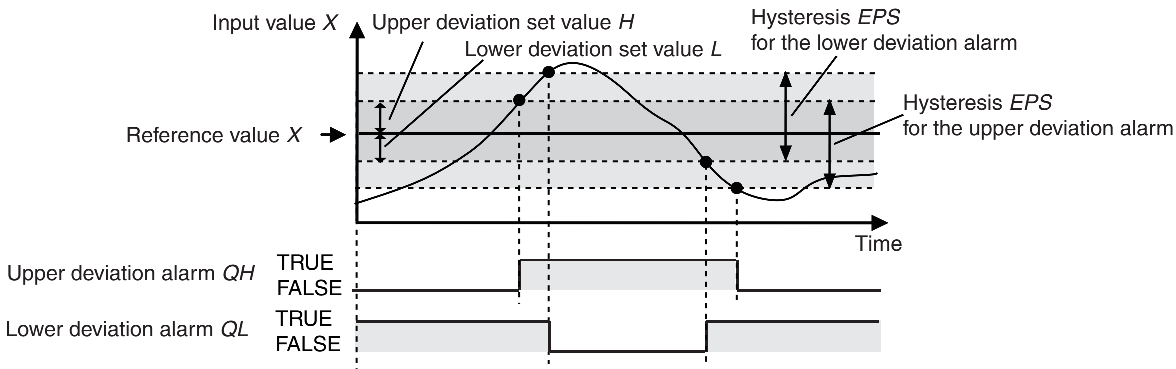
ST

```
LimitAlarmDv_LREAL_instance(A,PV,LREAL#50,SP,LREAL#40,LREAL#10,Alarm,H_Alarm,L_Alarm,Error0);
```

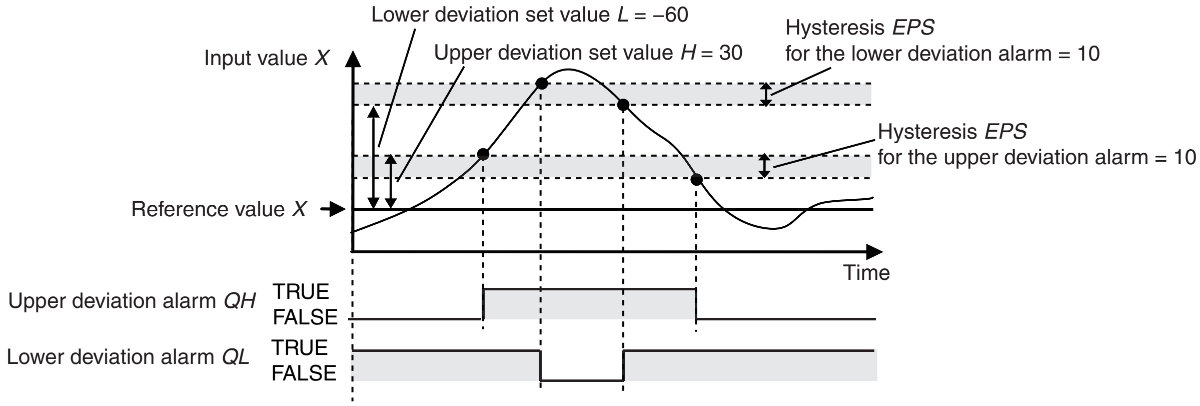


Additional Information

- Use the LimitAlarmDv_REAL instruction to reduce the instruction execution time.
- You can set EPS to less than *H* + *L*. If you do so, both *QH* and *QL* can be TRUE at the same time.



- You can set *H* and *L* so that *H* + *L* = 0. If you do so, either *QH* or *QL* will always be TRUE. For example, the following figure shows the operation when the value of *L* is –60 and the value of *H* is 30.



Precautions for Correct Use

- An error occurs if the value of *EPS* is outside of the valid range. *Error* changes to TRUE, and *Q*, *QH*, and *QL* change to FALSE.
- You can use this instruction for safety measures, for example, to turn OFF a temperature control output when a deviation alarm is output. If you do so, design the safety measures so that safety is maintained even when an error causes *Q*, *QH*, and *HL* to change to FALSE. For an application example, refer to the sample programming that is provided for the TimeProportionalOut instruction (page 2-733).

Version Information

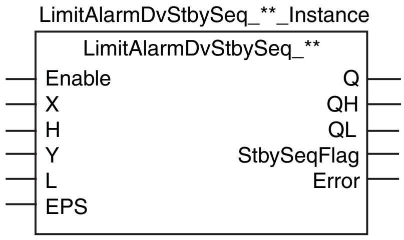
A CPU Unit with unit version 1.02 or later and Sysmac Studio version 1.03 or higher are required to use this instruction.

Sample Programming

Refer to the sample programming that is provided for the TimeProportionalOut instruction (page 2-733).

LimitAlarmDvStbySeq_**

The LimitAlarmDvStbySeq_** instruction outputs upper and lower deviation alarms with a standby sequence.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
LimitAlarmD-vStbySeq_**	Upper/Lower Deviation Alarm with Standby Sequence Group	FB	 “**” must be REAL or LREAL.	LimitAlarmDvStbySeq_**_instance(Enable, X, H, Y, L, EPS, Q, QH, QL, StbySeqFlag, Error); “**” must be REAL or LREAL.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Enable	Enable	Input	TRUE: Execute FALSE: Reset alarm	Depends on data type.	---	FALSE
X	Input value		Value for deviation alarm			
H	Upper deviation set value		Set value for an alarm for an upward deviation in respect to the reference value			0
Y	Reference value		Reference value for deviation			
L	Lower deviation set value		Set value for an alarm for a downward deviation in respect to the reference value			
EPS	Hysteresis		Hysteresis of the alarm	Depends on data type.*		
Q	Deviation alarm output	Output	TRUE: There is either an upper deviation alarm or a lower deviation alarm. FALSE: There is neither an upper deviation alarm nor a lower deviation alarm.	Depends on data type.	---	---
QH	Upper deviation alarm		TRUE: There is an upper deviation alarm. FALSE: There is no upper deviation alarm.			
QL	Lower deviation alarm		TRUE: There is a lower deviation alarm. FALSE: There is no lower deviation alarm.			
StbySeq Flag	Standby Sequence Enabled Flag		TRUE: Enabled FALSE: Disabled			

* Negative numbers are excluded.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Enable	OK																			
X														OK	OK					
H	Must be same data type as *X*.																			
Y	Must be same data type as *X*.																			
L	Must be same data type as *X*.																			
EPS	Must be same data type as *X*.																			
Q	OK																			
QH	OK																			
QL	OK																			
StbySeq Flag	OK																			

Function

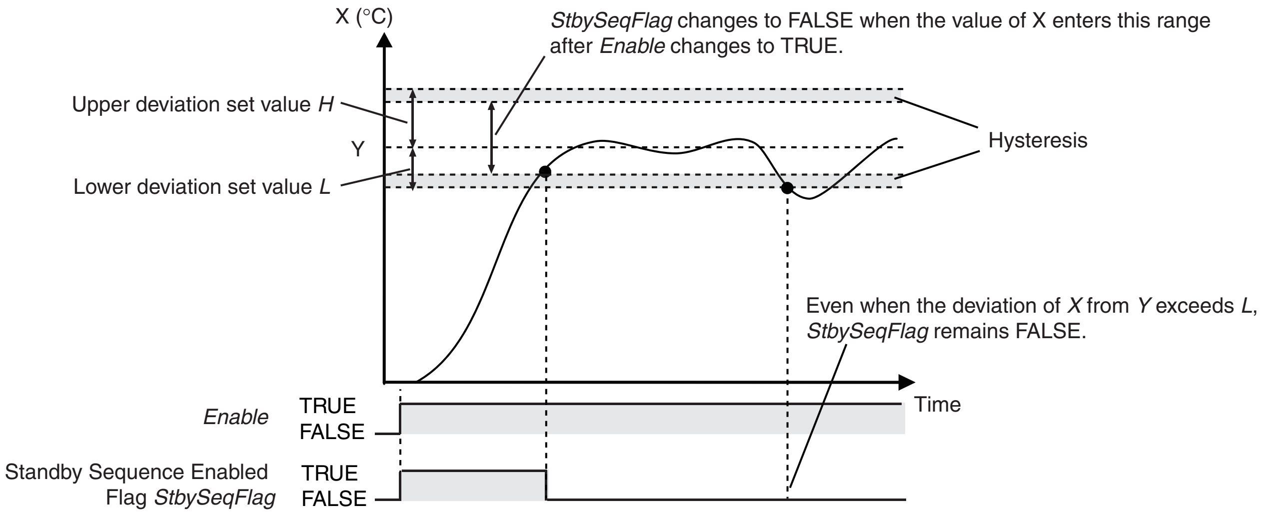
The LimitAlarmDvStbySeq_** instruction monitors the deviation in the input value from the reference value to see if it exceeds the lower deviation set value or the upper deviation set value. If the deviation exceeds the lower deviation set value or the upper deviation set value, the instruction outputs an alarm. However, the instruction will not output an alarm until the reference value first goes to between the lower and upper deviation set values. Use this instruction in temperature control, e.g., to not output a deviation alarm until the process temperature is stable.

The deviation in input value *X* from the reference value *Y* is monitored while *Enable* is TRUE. However, the deviation is not monitored while Standby Sequence Enabled Flag *StbySeqFlag* is TRUE. If the upper deviation in *X* from *Y* exceeds the value of upper deviation set value *H*, upper deviation alarm *QH* changes to TRUE. If the lower deviation in *X* from *Y* exceeds the value of lower deviation set value *L*, lower deviation alarm *QL* changes to TRUE. If the value of either *QH* or *QL* is TRUE, the value of alarm output *Q* is TRUE. The values of *X*, *H*, *Y*, *L*, and *EPS* are continuously updated while *Enable* is TRUE.

If *Enable* changes to FALSE, the alarm is reset. When the alarm is reset, *Q*, *QH*, *QL*, and *StbySeqFlag* change to FALSE.

StbySeqFlag changes to FALSE when all of the following conditions are met after *Enable* changes to TRUE. After *StbySeqFlag* changes to FALSE, it will not change to TRUE until *Enable* changes from FALSE to TRUE.

- Input value *X* – Reference value *Y* < Upper deviation set value *H* – Hysteresis *EPS*
- –(Input value *X* – Reference value *Y*) < Lower deviation set value *L* – Hysteresis *EPS*

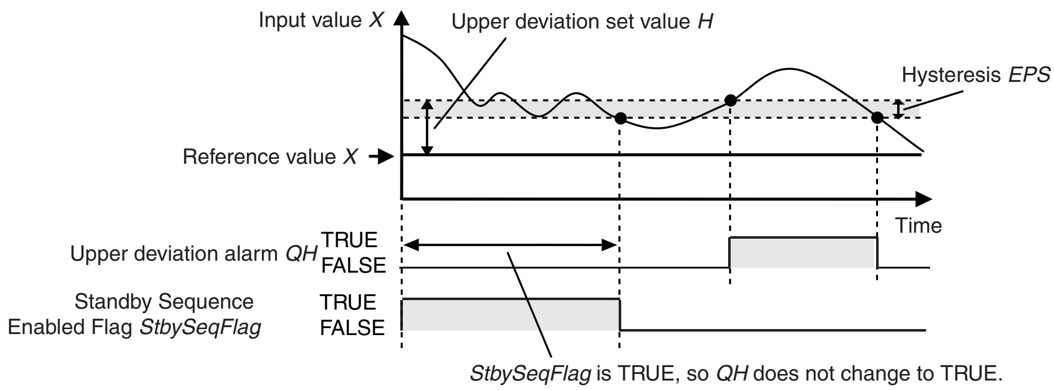


The data types of *X*, *H*, *Y*, *L*, and *EPS* must be either REAL or LREAL. The name of the instruction is determined by the data types of *X*, *H*, *Y*, *L*, and *EPS*. If the name of the instruction is LimitAlarmDvStby-Seq_LREAL, the data types of *X*, *H*, *Y*, *L*, and *EPS* are all LREAL.

Operation of Upper Deviation Alarm *QH*

Upper deviation alarm *QH* is the alarm for an upward deviation in respect to reference value *Y*. The value of *QH* changes as shown below while *StbySeqFlag* is FALSE. You can set the hysteresis to prevent hunting in the deviation alarm.

- If Input value *X* – Reference value *Y* > Upper deviation set value *H*, then *QH* is TRUE.
- If Input value *X* – Reference value *Y* < Upper deviation set value *H* – Hysteresis *EPS*, then *QH* is FALSE.

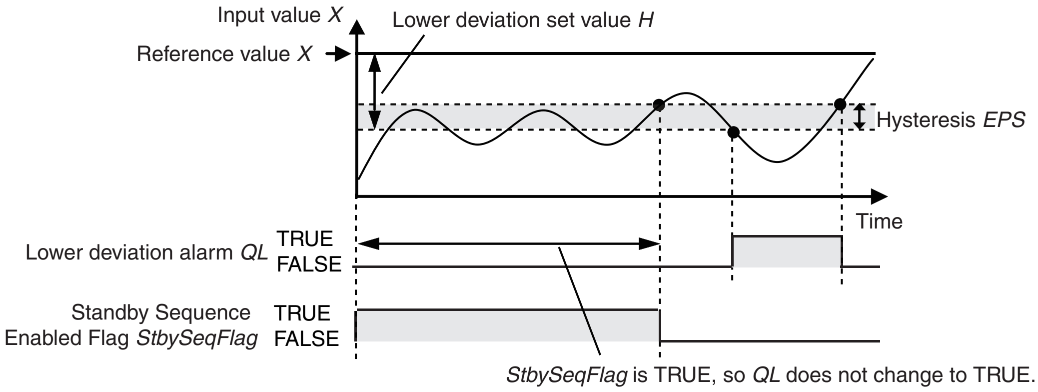


Operation of Lower Deviation Alarm *QL*

Lower deviation alarm *QL* is the alarm for a downward deviation in respect to reference value *Y*. The value of *QL* changes as shown below while *StbySeqFlag* is FALSE. You can set the hysteresis to prevent hunting in the deviation alarm.

- If –(Input value *X* – Reference value *Y*) > Lower deviation set value *L*, then *QL* is TRUE.

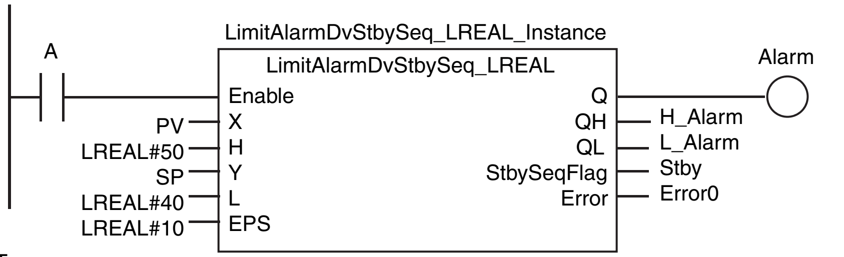
- If –(Input value *X* – Reference value *Y*) < Lower deviation set value *L* – Hysteresis *EPS*, then *QL* is FALSE.



Notation Example

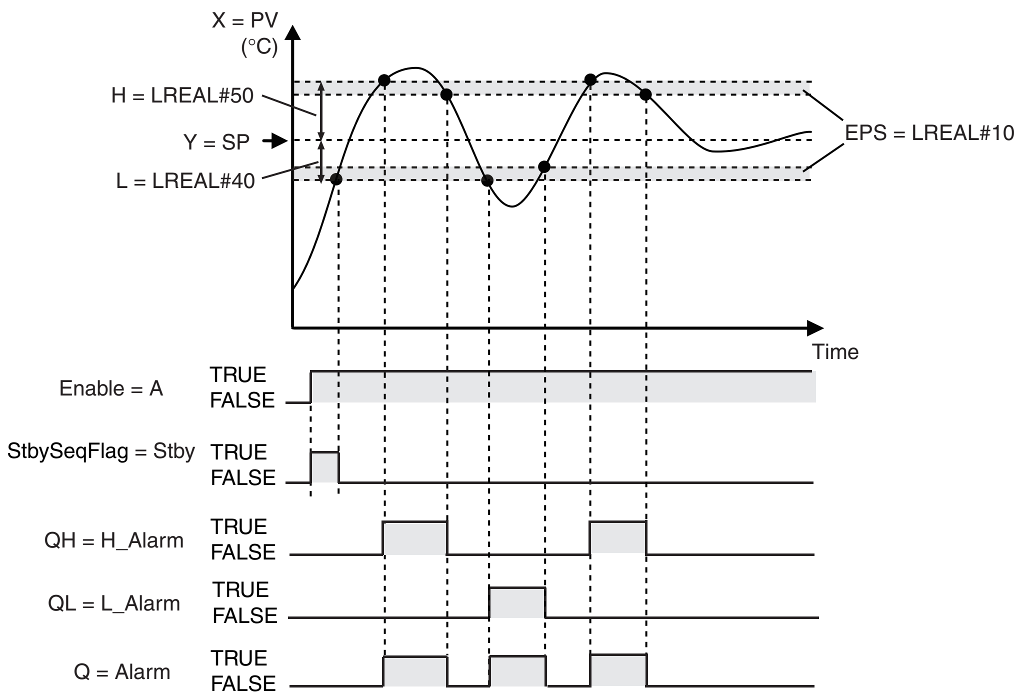
The following notation example sets upper deviation set value *H* to 50°C, lower deviation set value *L* to 40°C, and hysteresis *EPS* to 10°C.

LD



ST

```
LimitAlarmDvStbySeq_LREAL_Instance(A,PV,LREAL#50,SP,LREAL#40,LREAL#10,Alarm,H_Alarm,L_Alarm,Stby,Error0);
```



Additional Information

- Use the LimitAlarmDvStbySeq_REAL instruction to reduce the instruction execution time.
- You can set *EPS* to less than *H* + *L*. If you do so, both *QH* and *QL* can be TRUE at the same time. Refer to the *LimitAlarmDv_** instruction* (page 2-754).
- You can set *H* and *L* so that *H* + *L* < 0. If you do so, either *QH* or *QL* will always be TRUE while StbySeqFlag is FALSE. Refer to the *LimitAlarmDv_** instruction* (page 2-754).

Precautions for Correct Use

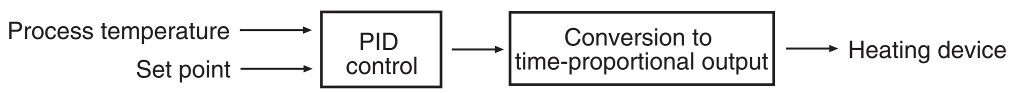
- An error occurs if the value of *EPS* is outside of the valid range. *Error* changes to TRUE, and *Q*, *QH*, and *QL* change to FALSE.
- You can use this instruction for safety measures, for example, to turn OFF a temperature control output when a deviation alarm is output. If you do so, design the safety measures so that safety is maintained even when an error causes *Q*, *QH*, and *QL* to change to FALSE. Refer to *Sample Programming* for an application example.

Version Information

A CPU Unit with unit version 1.02 or later and Sysmac Studio version 1.03 or higher are required to use this instruction.

Sample Programming

This sample performs temperature control for four points with upper/lower limit alarms and upper/lower deviation alarms with standby sequences. PID control is performed. The manipulated variables of PID control are converted to time-proportional output values that are output to heating devices.



Specifications

Temperature control is performed according to the following specifications.

Item	Specification
Input Unit	CJ1W-PH41U Isolated-type Universal Input Unit
Input types	K thermocouples
Output Unit	CJ1W-OD212 Transistor Output Unit
Set point	100°C
Upper limit of temperature	200°C
Lower limit of temperature	0°C
Hysteresis of upper/lower limit alarm	5°C
Upper deviation temperature	50°C
Lower deviation temperature	50°C
Hysteresis of upper/lower deviation alarm	3°C
Sampling period for PID control	100 ms
Output control period	1 s

Configuration and Settings

The following settings are used for the CJ1W-PH41U Input Unit.

Item	Set value
Input1:Input signal type	K(1)
Input2:Input signal type	K(1)
Input3:Input signal type	K(1)
Input4:Input signal type	K(1)

The following I/O map settings are used.

Unit	I/O port	Description	Variable
CJ1W-PH41U	Ch1_AIInPV	Process value for input 1 (INT data)	AI1
	Ch2_AIInPV	Process value for input 2 (INT data)	AI2
	Ch3_AIInPV	Process value for input 3 (INT data)	AI3
	Ch4_AIInPV	Process value for input 4 (INT data)	AI4
CJ1W-OD212	Ch1_Out00	Bit 00 of output word 1	DO1
	Ch1_Out01	Bit 01 of output word 1	DO2
	Ch1_Out02	Bit 02 of output word 1	DO3
	Ch1_Out03	Bit 03 of output word 1	DO4

The inputs and outputs for the temperature control for the four points correspond as shown below.

Input	Output
AI1	DO1
AI2	DO2
AI3	DO3
AI4	DO4

The task period of the task to which the program is assigned is 1 ms.

● Configuration Diagram

Refer to the sample programming that is provided for the TimeProportionalOut instruction (page 2-733).

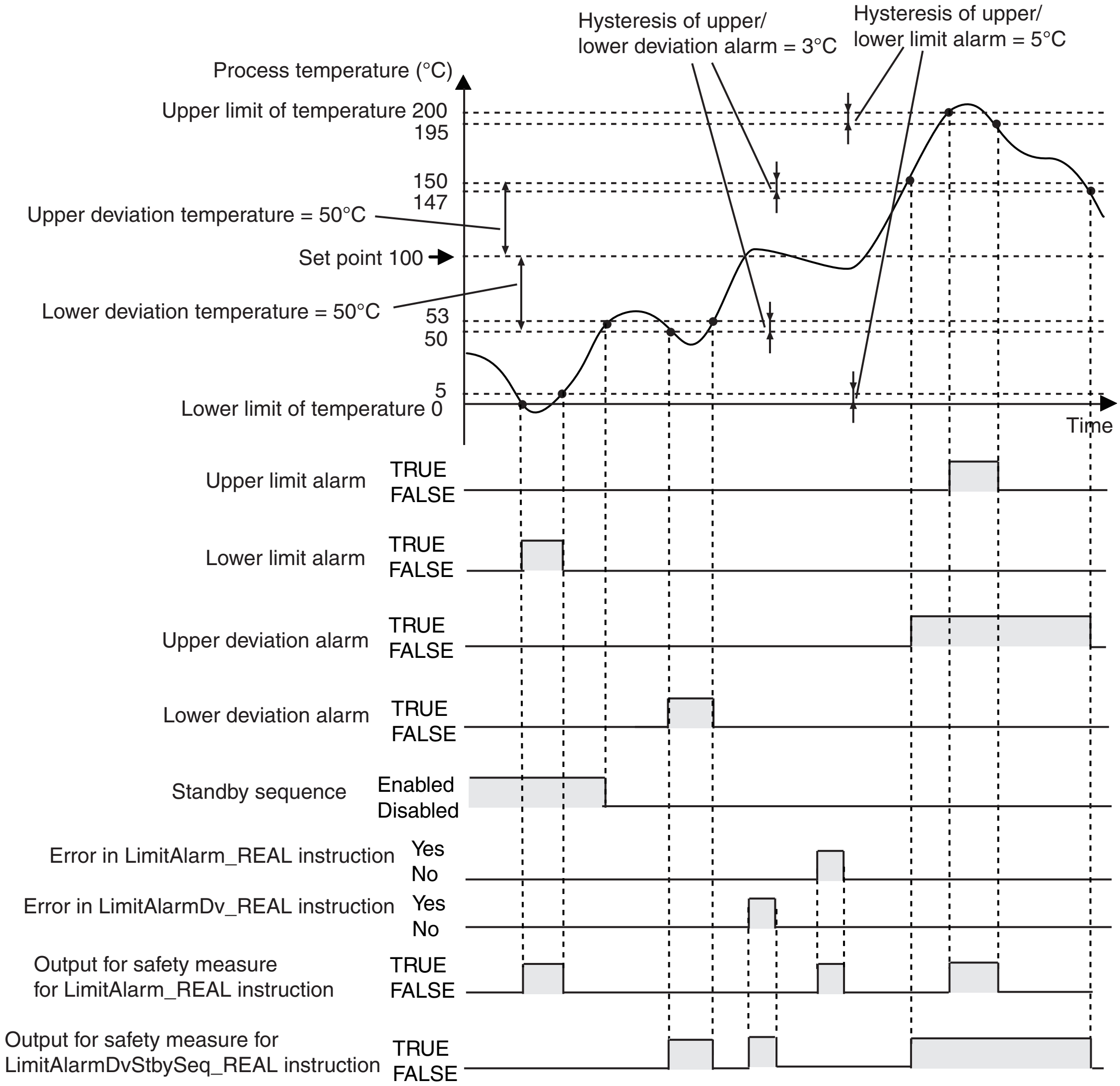
Processing

Perform the following procedure for all four points.

1. Get the process temperature.
2. Use the LimitAlarm_REAL instruction to output upper/lower limit alarms for the process temperature.
3. Perform an output as a safety measure if an error occurs in the LimitAlarm_REAL instruction or if an upper/lower limit alarm occurs.
4. Use the LimitAlarmDvStbySeq_REAL instruction to output upper/lower deviation alarms with a standby sequence for the deviation between the set point and the process temperature.
5. Perform an output as a safety measure if an error occurs in the LimitAlarmDvStbySeq_REAL instruction or if an upper/lower deviation alarm occurs.
6. Perform temperature control with the PIDAT instruction.

7 Use the TimeProportionalOut instruction to output the manipulated variable as a time-proportional value to the heating device.

● Operation of Upper/Lower Limit Alarms and Upper/Lower Deviation Alarms with Standby Sequence

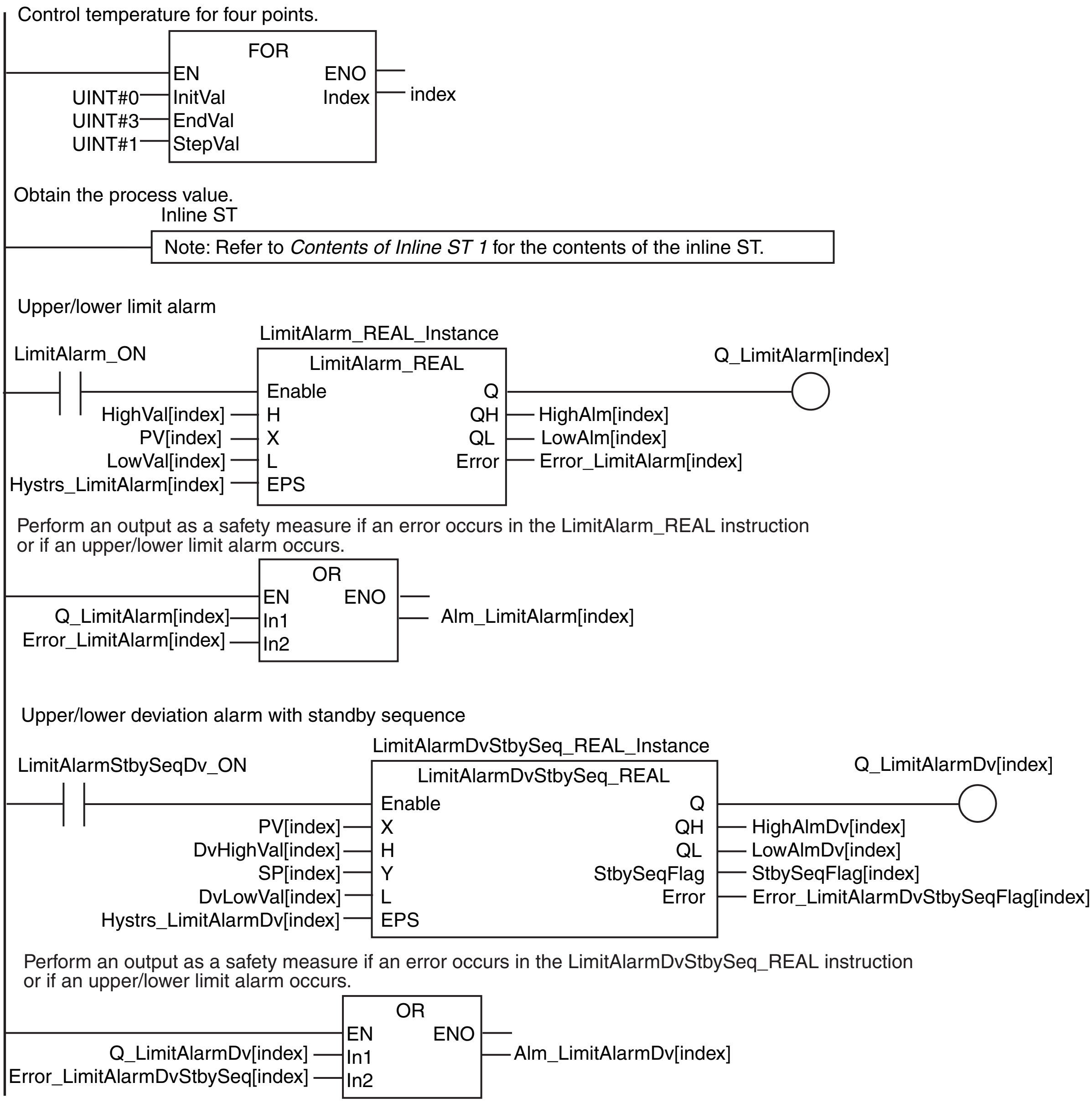


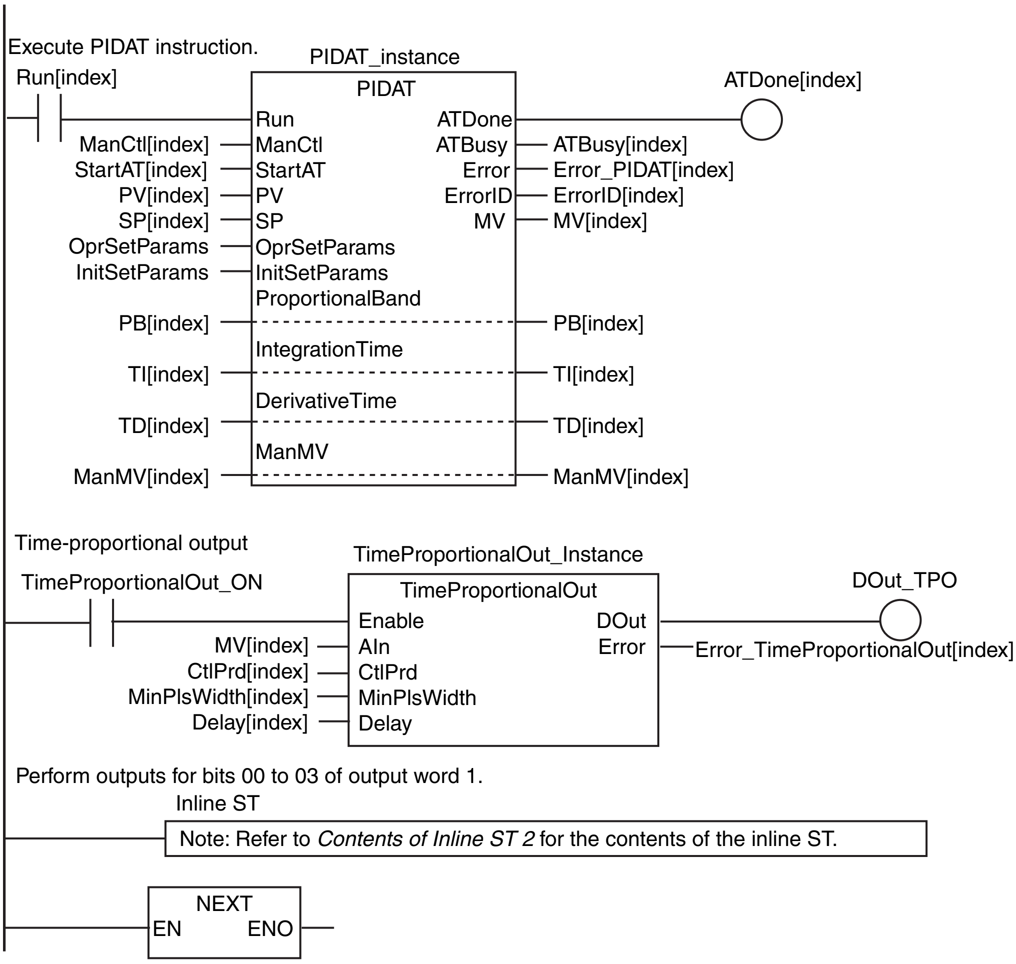
Application Programming

LD

Name	Data type	Default	Retain	Comment
index	UINT	0		Loop index
LimitAlarm_ON	BOOL	True		Execution of Upper/Lower Limit Alarm instruction
LimitAlarmDvStbySe-q_ON	BOOL	True		Execution of Upper/Lower Deviation Alarm with Standby Sequence instruction
TimeProportionalO ut_ON	BOOL	True		Execution of TimeProportionalOut instruction
AI	INT	0		Present value
PV	ARRAY[0..3] OF REAL	[4(0.0)]		Process value
SP	ARRAY[0..3] OF REAL	[4(100)]		Set point
DOut_TPO	BOOL	False		Time-proportional output
HighVal	ARRAY[0..3] OF REAL	[4(200)]		Upper limit set value of upper/lower limit alarm
LowVal	ARRAY[0..3] OF REAL	[4(0.0)]		Lower limit set value of upper/lower limit alarm
Hystrs_LimitAlarm	ARRAY[0..3] OF REAL	[4(5)]		Hysteresis of upper/lower limit alarm
Q_LimitAlarm	ARRAY[0..3] OF BOOL	[4(False)]		Upper/lower limit alarm output
HighAlm	ARRAY[0..3] OF BOOL	[4(False)]		Upper limit alarm
LowAlm	ARRAY[0..3] OF BOOL	[4(False)]		Lower limit alarm
Error_LimitAlarm	ARRAY[0..3] OF BOOL	[4(False)]		Error in LimitAlarm_-REAL instruction
Alm_LimitAlarm	ARRAY[0..3] OF BOOL	[4(False)]		Output for safety mea-sure for Upper/Lower Limit Alarm instruction
DvHighVal	ARRAY[0..3] OF REAL	[4(50)]		Upper deviation set value of upper/lower deviation alarm
DvLowVal	ARRAY[0..3] OF REAL	[4(50)]		Lower deviation set value of upper/lower deviation alarm
Q_LimitAlarmDv	ARRAY[0..3] OF BOOL	[4(False)]		Upper/lower deviation alarm output
HighAlmDv	ARRAY[0..3] OF BOOL	[4(False)]		Upper deviation alarm
LowAlmDv	ARRAY[0..3] OF BOOL	[4(False)]		Lower deviation alarm
StbySeqFlag	ARRAY[0..3] OF BOOL	[4(False)]		Standby Sequence Enabled Flag
Error_LimitAlarmD-vStbySeq	ARRAY[0..3] OF BOOL	[4(False)]		Error in LimitAlarmDvSt-bySeq_REAL instruc-tion
Hystrs_LimitAlarmDv	ARRAY[0..3] OF REAL	[4(3)]		Hysteresis of upper/lower deviation alarm

Name	Data type	Default	Retain	Comment
Alm_LimitAlarmDv	ARRAY[0..3] OF BOOL	[4(False)]		Output for safety measure for Upper/Lower Deviation Alarm instruction
Run	ARRAY[0..3] OF BOOL	[4(False)]		Execution condition
ManCtl	ARRAY[0..3] OF BOOL	[4(False)]		Manual/auto control
StartAT	ARRAY[0..3] OF BOOL	[4(False)]		Autotuning execution condition
OprSetParams	_sOPR_SET_PARAMS	(MVLowLmt:=0.0, MVUpLmt:=100.0, ManResetVal:=0.0, MVTrackSw:=False, MVTrackVal:=0.0, StopMV:=0.0, ErrorMV:=0.0, Alpha:=0.65, ATCalcGain:=1.0, ATHystrs:=0.2)		Operation setting parameters
InitSetParams	_sINIT_SET_PARAMS	(SampTime:=T#100ms, RngLow-Lmt:=–10.0, RngUpLmt:=1000.0, DirOpr:=False)		Initial setting parameters
PB	ARRAY[0..3] OF REAL	[4(10)]	✔	Proportional band
TI	ARRAY[0..3] OF TIME	[4(T#0S)]	✔	Integration time
TD	ARRAY[0..3] OF TIME	[4(T#0S)]	✔	Derivative time
ManMV	ARRAY[0..3] OF REAL	[4(0.0)]		Manual manipulated variable
ATDone	ARRAY[0..3] OF BOOL	[4(False)]		Autotuning normal completion
ATBusy	ARRAY[0..3] OF BOOL	[4(False)]		Autotuning busy
Error_PIDAT	ARRAY[0..3] OF BOOL	[4(False)]		Error in PIDAT instruction
ErrorID	ARRAY[0..3] OF WORD	[4(16#0)]		Error ID for PIDAT instruction
MV	ARRAY[0..3] OF REAL	[4(0.0)]		Manipulated variable
CtlPrd	ARRAY[0..3] OF TIME	[4(T#1s)]		Control period
MinPlsWidth	ARRAY[0..3] OF REAL	[4(0.0)]		Minimum pulse width
Delay	ARRAY[0..3] OF REAL	[4(0.0)]		ON-delay time
Error_TimeProportionalOut	ARRAY[0..3] OF BOOL	[4(False)]		Error in TimeProportionalOut instruction
LimitAlarm_REAL_instance	LimitAlarm_REAL			
LimitAlarmDvStbySeq_REAL_instance	LimitAlarmDvStbySeq_REAL			
PIDAT_instance	PIDAT			
TimeProportionalOut_instance	TimeProportionalOut			


Control temperature for four points.
FOR
EN
ENO
UINT#0
InitVal
Index
index
UINT#3
EndVal
UINT#1
StepVal
Obtain the process value.
Inline ST
Note: Refer to *Contents of Inline ST 1* for the contents of the inline ST.
Upper/lower limit alarm
LimitAlarm_REAL_Instance
LimitAlarm_ON
LimitAlarm_REAL
Q_LimitAlarm[index]
Enable
Q
HighVal[index]
H
QH
HighAlm[index]
PV[index]
X
QL
LowAlm[index]
LowVal[index]
L
Error
Error_LimitAlarm[index]
Hystrs_LimitAlarm[index]
EPS
Perform an output as a safety measure if an error occurs in the LimitAlarm_REAL instruction or if an upper/lower limit alarm occurs.
OR
EN
ENO
Q_LimitAlarm[index]
In1
Alm_LimitAlarm[index]
Error_LimitAlarm[index]
In2
Upper/lower deviation alarm with standby sequence
LimitAlarmDvStbySeq_REAL_Instance
LimitAlarmStbySeqDv_ON
LimitAlarmDvStbySeq_REAL
Q_LimitAlarmDv[index]
Enable
Q
PV[index]
X
QH
HighAlmDv[index]
DvHighVal[index]
H
QL
LowAlmDv[index]
SP[index]
Y
StbySeqFlag
StbySeqFlag[index]
DvLowVal[index]
L
Error
Error_LimitAlarmDvStbySeqFlag[index]
Hystrs_LimitAlarmDv[index]
EPS
Perform an output as a safety measure if an error occurs in the LimitAlarmDvStbySeq_REAL instruction or if an upper/lower limit alarm occurs.
OR
EN
ENO
Q_LimitAlarmDv[index]
In1
Alm_LimitAlarmDv[index]
Error_LimitAlarmDvStbySeq[index]
In2



● Contents of Inline ST 1

```
// Get values of inputs 1 to 4.
CASE index OF
  INT#0:
    AI:=AI1;
  INT#1:
    AI:=AI2;
  INT#2:
    AI:=AI3;
  ELSE
    AI:=AI4;
END_CASE;

// Convert PV AI to real number.
PV[index]:=INT_TO_REAL(AI)/REAL#10.0;
// CJ1W-PH41U output is ten times the process value, so divide by 10.0.
```

● Contents of Inline ST 2

```
// Perform outputs for bits 00 to 03 of output word 1.
CASE index OF
  INT#0:
    DO1:=DOut_TPO;
  INT#1:
    DO2:=DOut_TPO;
  INT#2:
    DO3:=DOut_TPO;
  ELSE
    DO4:=DOut_TPO;
END_CASE;
```

ST

Name	Data type	Default	Retain	Comment
index	UINT	0		Loop index
LimitAlarm_ON	BOOL	True		Execution of Upper/Lower Limit Alarm instruction
LimitAlarmDvStby-Seq_ON	BOOL	True		Execution of Upper/Lower Deviation Alarm with Standby Sequence instruction
TimeProportionalOut_ON	BOOL	True		Execution of Time-proportional Output instruction
AI	INT	0		Present value
PV	ARRAY[0..3] OF REAL	0.0		Process value
SP	ARRAY[0..3] OF REAL	[4(100)]		Set point
DOut_TPO	BOOL	False		Time-proportional output
HighVal	ARRAY[0..3] OF REAL	[4(200)]		Upper limit set value of upper/lower limit alarm
LowVal	ARRAY[0..3] OF REAL	[4(0.0)]		Lower limit set value of upper/lower limit alarm
Hystrs_LimitAlarm	ARRAY[0..3] OF REAL	[4(5)]		Hysteresis of upper/lower limit alarm
Q_LimitAlarm	ARRAY[0..3] OF BOOL	[4(False)]		Upper/lower limit alarm output
HighAlm	ARRAY[0..3] OF BOOL	[4(False)]		Upper limit alarm
LowAlm	ARRAY[0..3] OF BOOL	[4(False)]		Lower limit alarm
Error_LimitAlarm	ARRAY[0..3] OF BOOL	[4(False)]		Error in LimitAlarm_-REAL instruction
Alm_LimitAlarm	ARRAY[0..3] OF BOOL	[4(False)]		Output for safety measure for Upper/Lower Limit Alarm instruction
DvHighVal	ARRAY[0..3] OF REAL	[4(50)]		Upper deviation set value of upper/lower deviation alarm
DvLowVal	ARRAY[0..3] OF REAL	[4(50)]		Lower deviation set value of upper/lower deviation alarm
Q_LimitAlarmDv	ARRAY[0..3] OF BOOL	[4(False)]		Upper/lower deviation alarm output
HighAlmDv	ARRAY[0..3] OF BOOL	[4(False)]		Upper deviation alarm
LowAlmDv	ARRAY[0..3] OF BOOL	[4(False)]		Lower deviation alarm
StbySeqFlag	ARRAY[0..3] OF BOOL	[4(False)]		Standby Sequence Enabled Flag
Error_LimitAlarmD-vStbySeq	ARRAY[0..3] OF BOOL	[4(False)]		Error in LimitAlarmD-vStbySeq_REAL instruction
Hystrs_LimitAlarmDv	ARRAY[0..3] OF REAL	[4(3)]		Hysteresis of upper/lower deviation alarm
Alm_LimitAlarmDv	ARRAY[0..3] OF BOOL	[4(False)]		Output for safety measure for Upper/Lower Deviation Alarm instruction
Run	ARRAY[0..3] OF BOOL	[4(False)]		Execution condition
ManCtl	ARRAY[0..3] OF BOOL	[4(False)]		Manual/auto control

Name	Data type	Default	Retain	Comment
StartAT	ARRAY[0..3] OF BOOL	[4(False)]		Autotuning execution condition
OprSetParams	_sOPR_SET_PARAMS	(MVLowLmt:=0.0, MVUpLmt:=100.0, ManResetVal:=0.0, MVTrackSw:=False, MVTrackVal:=0.0, StopMV:=0.0, ErrorMV:=0.0, Alpha:=0.65, ATCalcGain:=1.0, ATHystrs:=0.2)		Operation setting parameters
InitSetParams	_sINIT_SET_PARAMS	(SampTime:=T#100ms, RngLowmt:=–10.0, RngUpLmt:=1000.0, DirOpr:=False)		Initial setting parameters
PB	ARRAY[0..3] OF REAL	[4(10)]	✔	Proportional band
TI	ARRAY[0..3] OF TIME	[4(T#0S)]	✔	Integration time
TD	ARRAY[0..3] OF TIME	[4(T#0S)]	✔	Derivative time
ManMV	ARRAY[0..3] OF REAL	[4(0.0)]		Manual manipulated variable
ATDone	ARRAY[0..3] OF BOOL	[4(False)]		Autotuning normal completion
ATBusy	ARRAY[0..3] OF BOOL	[4(False)]		Autotuning busy
Error_PIDAT	ARRAY[0..3] OF BOOL	[4(False)]		Error in PIDAT instruction
ErrorID	ARRAY[0..3] OF WORD	[4(16#0)]		Error ID for PIDAT instruction
MV	ARRAY[0..3] OF REAL	[4(0.0)]		Manipulated variable
CtlPrd	ARRAY[0..3] OF TIME	[4(T#1s)]		Control period
MinPlsWidth	ARRAY[0..3] OF REAL	[4(0.0)]		Minimum pulse width
Delay	ARRAY[0..3] OF REAL	[4(0.0)]		ON-delay time
Error_TimeProportionalOut	ARRAY[0..3] OF BOOL	[4(False)]		Error in TimeProportionalOut instruction
LimitAlarm_REAL_instance	LimitAlarm_REAL			
LimitAlarmDvStbySeq_REAL_instance	LimitAlarmDvStbySeq_REAL			
PIDAT_instance	PIDAT			
TimeProportionalOut_instance	TimeProportionalOut			

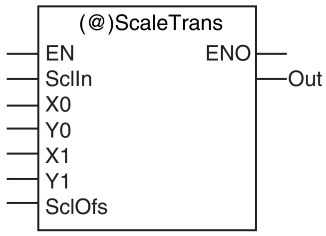
```
// Control temperature for four points.
FOR index:=UINT#0 TO UINT#3 BY UINT#1 DO

    // Get values of inputs 1 to 4.
    CASE index OF
      INT#0:
        AI:=AI1;
      INT#1:
        AI:=AI2;
      INT#2:
        AI:=AI3;
      ELSE
        AI:=AI4;
    END_CASE;

  // Convert PV AI to real number.
  PV[index]:=INT_TO_REAL(AI)/REAL#10.0; // CJ1W-PH41U output is ten times the
  // process value, so divide by 10.0.

  // Upper/lower limit alarm
```

```
  LimitAlarm_REAL_instance(
    Enable :=LimitAlarm_ON,
    H      :=HighVal[index],
    X      :=PV[index],
    L      :=LowVal[index],
    EPS    :=Hystrs_LimitAlarm[index],
    Q      =>Q_LimitAlarm[index],
    QH     =>HighAlm[index],
    QL     =>LowAlm[index],
    Error  =>Error_LimitAlarm[index]);

// Perform an output as a safety measure if an error occurs in the
// LimitAlarm_REAL instruction or if an upper/lower limit alarm occurs.
  Alm_LimitAlarm[index]:=Q_LimitAlarm[index] OR Error_LimitAlarm[index];

  // Upper/lower deviation alarm with standby sequence
  LimitAlarmDvStbySeq_REAL_instance(
    Enable      :=LimitAlarmDvStbySeq_ON,
    X           :=PV[index],
    H           :=DvHighVal[index],
    Y           :=SP[index],
    L           :=DvLowVal[index],
    EPS         :=Hystrs_LimitAlarmDv[index],
    Q            =>Q_LimitAlarmDv[index],
    QH           =>HighAlmDv[index],
    QL           =>LowAlmDv[index],
    StbySeqFlag =>StbySeqFlag[index],
    Error        =>Error_LimitAlarmDvStbySeq[index]);

  // Perform an output as a safety measure if an error occurs in the
  // LimitAlarmDvStbySeq_REAL instruction
  // or if an upper/lower limit alarm occurs.
  Alm_LimitAlarmDv[index]:=Q_LimitAlarmDv[index] OR
  Error_LimitAlarmDvStbySeq[index];

  // Execute PIDAT instruction.
  PIDAT_instance(
    Run              :=Run[index],
    ManCtl           :=ManCtl[index],
    StartAT          :=StartAT[index],
    PV               :=PV[index],
    SP               :=SP[index],
    OprSetParams     :=OprSetParams,
    InitSetParams    :=InitSetParams,
    ProportionalBand:=PB[index],
    IntegrationTime :=TI[index],
    DerivativeTime  :=TD[index],
    ManMV            :=ManMV[index],
    ATDone            =>ATDone[index],
    ATBusy            =>ATBusy[index],
    Error             =>Error_PIDAT[index],
    ErrorID           =>ErrorID[index],
    MV                =>MV[index]);

  // Time-proportional output
  TimeProportionalOut_instance(
    Enable      :=TimeProportionalOut_ON,
    AIn         :=MV[index],
    CtlPrd      :=CtlPrd[index],
    MinPlsWidth :=MinPlsWidth[index],
    Delay       :=Delay[index],
    DOut         =>DOut_TPO,
    Error        =>Error_TimeProportionalOut[index]);

  // Perform outputs for bits 00 to 03 of output word 1.
```

```
  CASE index OF
    INT#0:
      DO1:=DOut_TPO;
    INT#1:
      DO2:=DOut_TPO;
    INT#2:
      DO3:=DOut_TPO;
    ELSE
      DO4:=DOut_TPO;
  END_CASE;

END_FOR;
```

ScaleTrans

The ScaleTrans instruction converts input values from an input range to an output range.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ScaleTrans	Scale Transformation	FUN		Out :=ScaleTrans(SclIn,X0,Y0,X1,Y1, SclOfs);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
SclIn	Input value	Input	Value to scale	Depends on data type.	---	*
X0	Input range lower limit		Lower limit of input range			0
Y0	Output range lower limit		Lower limit of output range			
X1	Input range upper limit		Upper limit of input range			
Y1	Output range upper limit		Upper limit of output range			
SclOfs	Offset		Offset for output value			
Out	Output Value	Output	Value after scale transformation		---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
SclIn														OK	OK					
X0														OK	OK					
X1														OK	OK					
Y0														OK	OK					
Y1														OK	OK					
SclOfs														OK	OK					
Out														OK	OK					

Function

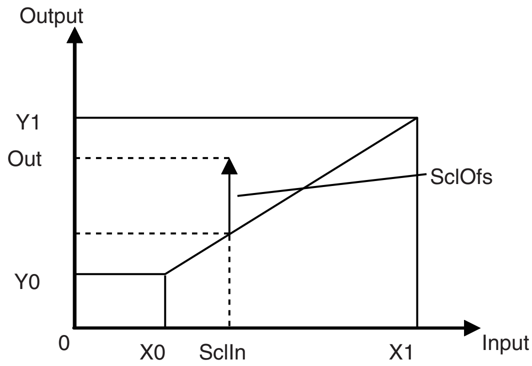
The Scale Trans instruction scales the value of input value *SclIn* from an input range to an output range.

The input range is specified with input range lower limit *X0* and input range upper limit *X1*. The output range is specified with output range lower limit *Y0* and output range upper limit *Y1*.

The value of offset *SclOfs* is added to the value that was scaled to the output range and the result is output as output value *Out*. *SclOfs* is used, for example, to correct for error in temperature control.

The following conversion is used.

$$Out = \frac{Y1-Y0}{X1-X0}(SclIn - X0) + Y0 + SclOfs$$



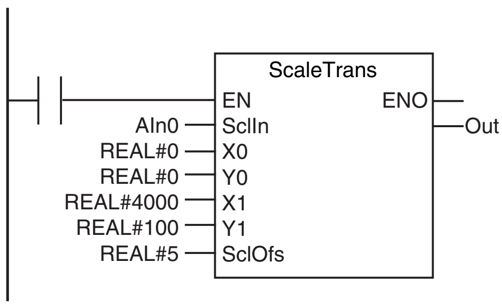
Notation Example

The following notation example scales an input value of 2,500 from an input range of 0 to 4,000 to an output range of 0% to 100%. An offset of 5% is added to the output value.

The following values are used: *SclIn* = REAL#2500, *X0* = REAL#0, *X1* = REAL#4000, *Y0* = REAL#0, *Y1* = REAL#100, and *SclOfs* = REAL#5.

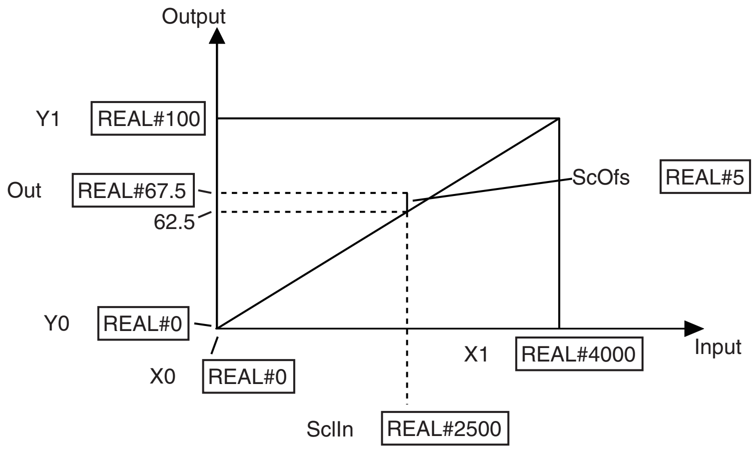
The value of *Out* will be REAL#67.5.

LD



ST

```
Out :=ScaleTrans(AIn0,REAL#0,REAL#0,REAL#4000,
                REAL#100,REAL#5);
```



An input value of 2,500 is scaled to 62.5 for an input range of 0 to 4,000 and an output range of 0 to 100. When an offset of 5 is added, *Out* becomes REAL#67.5.

Additional Information

- When scaling *SclIn* to the ranges of the values of *PV* and *SP* of the PIDAT instruction, pass the following parameters to *Y0* and *Y1*.

Variable	Parameter
Y0	InitSetParams.RngLowLmt (input range lower limit of the PIDAT instruction)
Y1	InitSetParams.RngUpLmt (input range upper limit of the PIDAT instruction)

- Settings are also possible with *X1* < *X0* and *Y1* < *Y0*.

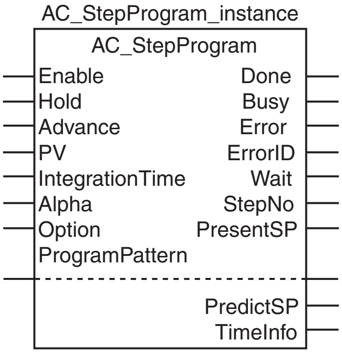
Precautions for Correct Use

Version Information

A CPU Unit with unit version 1.05 or later and Sysmac Studio version 1.06 or higher are required to use this instruction.

AC_StepProgram

The AC_StepProgram instruction calculates the present set point and the predicted set point every task period according to the specified program pattern.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
AC_Step Program	Step Program	FB		AC_StepProgram_instance(Enable, Hold, Advance, PV, IntegrationTime, Alpha, Option, ProgramPattern, Done, Busy, Error, ErrorID, Wait, StepNo, PresentSP, PredictSP, TimeInfo);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Enable	Enable	Input	TRUE: Execute FALSE: Stop	Depends on data type.	---	FALSE
Hold	Hold		TRUE: Hold FALSE: Do not hold			
Advance	Advance		The number of the step that is executed is incremented each time this variable changes to TRUE.			
PV	Process value		Measured value (process value)*1			0
Integration-Time	Integration time		Integration time*2	T#0.0000s to T#10000.0000s*3	s	T#0s
Alpha	2-PID parameter α		2-PID parameter α*4	0.00 to 1.00	---	0
Option	Option		Option*5	---		---
ProgramPat-tern[] array	Program pattern	In-out	Program pattern	---	---	---
Wait	Waiting	Output	TRUE: Waiting FALSE: Not waiting	Depends on data type.	---	---
StepNo	Present step number		The number of the current step	0 to 99		
PresentSP	Present set point		The calculated present set point	Depends on data type.		
PredictSP	Predicted set point		The calculated predicted set point			
TimeInfo	Clock information		Clock information to monitor the progress of the instruction	---		

*1 It is the same as *PV* in the PIDAT instruction. Refer to *PV (Process Value)* on page 2-781 for details.

*2 It is the same as *IntegrationTime* in the PIDAT instruction. Refer to *IntegrationTime (Integration Time)* on page 2-781 for details.

*3 Digits below 0.0001 s are truncated.

*4 It is the same as *OprSetParams.Alpha* in the PIDAT instruction. Refer to *Alpha (2-PID Parameter α)* on page 2-782 for details.

*5 Refer to *Structure Specifications* on page 2-780 for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Enable	OK																			
Hold	OK																			
Advance	OK																			
PV														OK						
Integration-Time																OK				
Alpha														OK						
Option	Refer to *Function* for details on the structure _sAC_STEP_OPTION.																			
ProgramPattern[] array*1*2*3	Refer to *Function* for details on the structure _sAC_STEP_DATA. Specify an array.																			
Wait	OK																			
StepNo						OK														
PresentSP														OK						
PredictSP														OK						
TimeInfo	Refer to *Function* for details on the structure _sAC_STEP_TIME.																			

*1 The array can have a maximum of 100 elements.

*2 This is a one-dimensional array. If an array with more than one dimension is specified, a building error will occur.

*3 The first array element number is 0. If a number other than 0 is specified for the first array element, a building error will occur.

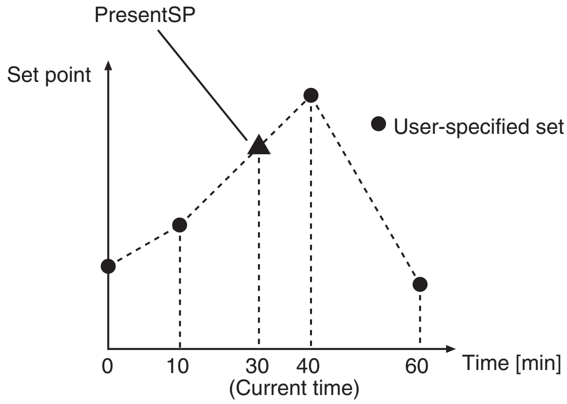
Function

You use the AC_StepProgram instruction together with the PIDAT instruction to calculate present set point *PresentSP* and predicted set point *PredictSP* every task period when you perform manipulated variable control for a temperature controller or other controller.

The present set point is the set point in the present task period. The predicted set point is arrived at by applying delay compensation for 2-PID control to the present set point. By passing predicted set point *PredictSP* to set point *SP* of the PIDAT instruction, you can improve the tracking characteristic of programmed control with the PIDAT instruction.

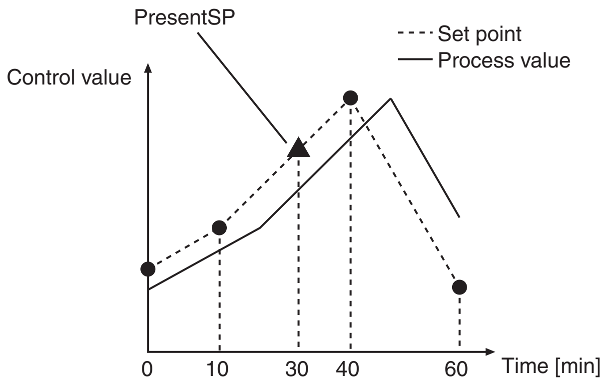
PresentSP (Present Set Point)

Present set point *PresentSP* is the set point in the present task period. For example, assume that the user sets the set points for 0, 10, 40, and 60 minutes after the start of control as shown below. Also assume that the current time is 30 minutes after the start of control. The AC_StepProgram instruction performs linear interpolation of the set points for 10 minutes and 40 minutes after the start of control and calculates *PresentSP*.

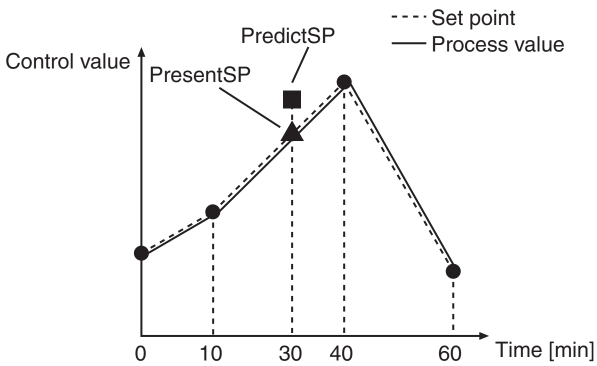


PredictSP (Predicted Set Point)

Predicted set point *PredictSP* is arrived at by applying delay compensation for 2-PID control to present set point *PresentSP*. If *PresentSP* is passed to *SP* in the PIDAT instruction without compensation, *PV* in the PIDAT instruction will not match the set point. This is illustrated in the following figure.



The set point that can been compensated for delay is given by *PredictSP*. The AC_StepProgram instruction calculates *PredictSP* based on integration time *IntegrationTime* and 2-PID parameter α *Alpha*. By passing *PredictSP* to *SP* of the PIDAT instruction, the tracking characteristic of programmed control with the PIDAT instruction is improved.



Structure Specifications

The data type of *Option* is structure _sAC_STEP_OPTION. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Option	Option	Option	_sAC_STEP _OPTION	---	---	---
StartAtPV	Start at PV	TRUE: Enable starting at PV FALSE: Disable starting at PV	BOOL	Depends on data type.		FALSE
StartStepNo	Start step number	The step number from which to start processing	USINT	0 to 99		0
EndStepNo	End step number	The step number from which to end processing*1	USINT			
Reserved	Reserved.	Reserved.	ARRAY[0..31] OF BYTE	Depends on data type.		All 32 elements contain 0.

*1 A setting of 0 treats the highest element number in *ProgramPattern[]* as the end step number.

The data type of the elements of program pattern *ProgramPattern[]* is structure _sAC_STEP_DATA. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Program Pattern	Program pattern	Program pattern	_sAC_STEP _DATA	---	---	---
ReachSP	Target set point	The target step point for the step	REAL	Depends on data type.		0
TimeWidth	Time width	The time width of the step*1	TIME		s	T#0s
WaitWidth	Wait width	The wait width of the step*2	REAL		---	0
WaitTime Limit	Wait time upper limit	The upper limit of the wait width of the step*1*3	TIME		s	T#0s

*1 The resolution is one task period.

*2 A setting of 0 or less is treated as 0.

*3 A setting of 0 or less is treated as T#0s.

The data type of clock information *TimeInfo* is structure _sAC_STEP_TIME. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
TimeInfo	Clock information	Clock information	_sAC_STEP_TIME	---	---	---
ProgramTime	Program time	The total of *TimeWidth* from step 0 to *EndStepNo*.	TIME	Non-negative value	s	T#0s
ElapseTime	Elapsed time	The elapsed time from when instruction execution started*1	TIME			
ProgressTime	Progress time	The elapsed time from when instruction execution started*2	TIME			
LeftTime	Remaining time	The time from the present until all processing is completed*2	TIME			
StepProgressTime	Step progress time	The elapsed time from the start of the current step*2	TIME			
StepLeftTime	Step remaining time	The time from the present until all processing is completed for the current step*2	TIME			

*1 Includes the wait time. Does not include the hold time.

*2 This value does not include the wait time and hold time.

Meanings of Variables

The meanings of the variables that are used in this instruction are described below.

● *Enable* (Enable)

This is the execution condition for the instruction.

Instruction execution starts when *Enable* changes to TRUE. Instruction execution stops when *Enable* changes to FALSE.

● *Hold* (Hold)

This is the execution flag for holding.

Holding is performed when *Hold* changes to TRUE.

Details on holding are provided later.

● *Advance* (Advance)

If the value changes to TRUE during instruction execution, processing moves to the next step.

Details on advancing are provided later.

● *PV* (Process Value)

This variable gives the process value of the controlled system. It is the same as *PV* in the PIDAT instruction.

● *IntegrationTime* (Integration Time)

This variable is the same as *IntegrationTime* in the PIDAT instruction.

Input the value of *IntegrationTime* or the *IntegrationTime* variable in the PIDAT instruction or PIDAT_HeatCool instruction.

- ***Alpha* (2-PID Parameter α)**

This variable is the same as *OprSetParams.Alpha* in the PIDAT instruction.

Input the value of *OprSetParams.Alpha* or the *OprSetParams.Alpha* variable in the PIDAT instruction or PIDAT_HeatCool instruction.

- ***StartAtPV* (Start at PV)**

This variable is the execution flag for starting at the process value.

Starting at the process value is performed when *StartAtPV* is TRUE.

Details on starting at the process value are provided later.

- ***StartStepNo* (Start Step Number) and *EndStepNo* (End Step Number)**

These variables give the number for the step from which to start processing and the number of the step to end processing of the steps in the program pattern.

A setting of 0 for *EndStepNo* treats the highest element number in *ProgramPattern[]* as the end step number.

Details on program patterns and steps are provided later.

- ***ReachSP* (Target Set Point)**

This variable gives the set point that should be reached at the end of the step in the program pattern.

Details on program patterns and steps are provided later.

- ***TimeWidth* (Time Width)**

This variable gives the time width for the step in the program pattern.

Details on program patterns and steps are provided later.

- ***WaitWidth* (Wait Width)**

This variable gives the threshold for performing waiting in the step in the program pattern.

Details on waiting are provided later.

- ***WaitTimeLimit* (Wait Time Limit)**

This variable gives the upper limit of the wait time for waiting in the step in the program pattern.

If the value of *WaitTimeLimit* is T#0, the upper limit of the wait time is infinity.

Details on waiting are provided later.

- ***Wait* (Waiting)**

This variable is a flag that indicates if waiting is in progress.

If *Wait* is TRUE, waiting is in progress.

Details on waiting are provided later.

- ***StepNo* (Present Step Number)**

This variable gives the number of the current step.

Details on program patterns and steps are provided later.

- ***PresentSP* (Present Set Point)**

This variable gives the calculated present set point.

- ***PredictSP* (Predicted Set Point)**

This variable gives the calculated predicted set point.

- ***ProgramTime* (Program Time)**

This variable gives the total of *TimeWidth* from step 0 to *EndStepNo* in the program pattern.

Details on program patterns and steps are provided later.

- ***ElapseTime* (Elapsed Time)**

This variable gives the elapsed time from when instruction execution started. This value includes the wait time but not the hold time.

Details on waiting and holding are provided later.

- ***ProgressTime* (Progress Time)**

This variable gives the elapsed time from when instruction execution started. This value does not include the wait time and hold time.

Details on waiting and holding are provided later.

- ***LeftTime* (Remaining Time)**

This variable gives the time from the present until all processing is completed. This value does not include the wait time and hold time.

Details on waiting and holding are provided later.

- ***StepProgressTime* (Step Progress Time)**

This variable gives the elapsed time from the start of the current step in the program pattern. This value does not include the wait time and hold time.

Details on program patterns, steps, waiting, and holding are provided later.

- ***StepLeftTime* (Step Remaining Time)**

This variable gives the time from the present until all processing is completed for the current step in the program pattern. This value does not include the wait time and hold time.

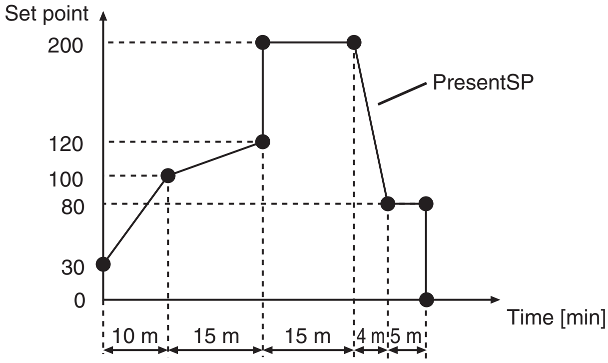
Details on program patterns, steps, waiting, and holding are provided later.

Program Pattern

The program pattern divides the processing from the start to end of execution of the instruction into steps and chronologically gives the target set point and time width for each step. The program pattern is expressed in the *ProgramPattern[]* array, which has elements with a data type of _sAC_STEP_DATA. Each element of *ProgramPattern[]* corresponds to one step.

An example of a program pattern is provided below. If the values of the *ReachSP* and *TimeWidth* elements of *ProgramPattern[]* are as given in the following table, the relation between time and the set points after instruction execution is started is shown in the following figure.

	ProgramPattern[] element number							
	0	1	2	3	4	5	6	7
Step number	0	1	2	3	4	5	6	7
Value of *ReachSP*	30	100	120	200	200	80	80	0
Value of *TimeWidth*	T#0s	T#10m	T#15m	T#0s	T#15m	T#4m	T#5m	T#0s



Linear interpolation is perform for the set points for the steps and the value of *PresentSP* is calculated for each point. The solid line in the figure represents *PresentSP*. For each task period, the value of *PresentSP* at that point is output.

● Relation between the Value of *TimeWidth* and the Time Width of the Step

The following table shows the relation between the value of *TimeWidth* and the time width of the step.

Value of *TimeWidth*	Step number	Time width of the step
T#0s	0	Treated as T#0s.
	Not 0	Treated as one task period.
Positive	---	The value of *TimeWidth* is the time width of the step.
Negative	---	Treated as one task period.

● Operation for Step Time Width That Is Less Than One Task Period

The resolution of the step time width is one task period. The following table describes the operation for a step time width that is less than one task period.

Step number	Time width of the step	Operation
0	T#0s	The value of *ReachSP* for step 0 is the initial value for *PresentSP*. Actual processing starts from step 1.
	Not T#0s	Processing for the current step is executed for only one task period and then processing moves to the next step.
Not 0	---	

Start Step Number *StartStepNo* and End Step Number *EndStepNo*

You can set any steps in the program pattern as the start step and the end step for processing. Set the number of the start step in *StartStepNo* and the number of the end step in *EndStepNo.*

For example, if you set *StartStepNo* to 3 and *EndStepNo* to 6 when you execute the instruction, processing is performed from step 3 through step 6.

● Changing the Value of *StartStepNo* or *EndStepNo* during Instruction Execution

You can change the values of *StartStepNo* and *EndStepNo* during execution of the instruction. The operation that occurs if you change these values is described in the following table.

Variable	New step number	Operation
StartStepNo	---	Processing will start from the beginning of the step specified by the new *StartStepNo*.
EndStepNo	Changing to a step number that is equal to or higher than the current step number	Progressing will end when the step specified by the new *EndStepNo* is completed.
	Changing to a step number that is lower than the current step number	Processing ends as soon as the end step number is changed. The value of *Done* changes to TRUE.

Waiting

Due to delays in the controlled system, the value of *PV* may not reach the value of *ReachSP* with *TimeWidth* for the current step. Waiting can be applied to continue processing the current step even if the time width specified in *TimeWidth* is exceeded.

The following variables in *ProgramPattern[]* are related to waiting: wait width *WaitWidth*, wait time upper limit *WaitTimeLimit*, and waiting *Wait*.

● Condition for Waiting

Waiting occurs if the difference between *ReachSP* and *PV* exceeds *WaitWidth* after the end time for the current step.

● End of Waiting

If the difference between *ReachSP* and *PV* becomes equal to or less than *WaitWidth* before *WaitTimeLimit* is reached after the start of waiting, waiting ends at that time and processing moves to the next step.

If the difference between *ReachSP* and *PV* does not become equal to or less than *WaitWidth* before *WaitTimeLimit* is reached after the start of waiting, waiting ends when the time set for *WaitTimeLimit* expires and processing moves to the next step. However, if the value of *WaitTimeLimit* is T#0, the upper limit of the wait time is infinity. Therefore, waiting occurs without a time limit until the difference between *ReachSP* and *PV* becomes less than or equal to *WaitWidth*.

● Monitoring Waiting

You can monitor waiting with the value of *Wait*. If processing is currently waiting, the value of *Wait* is TRUE. If processing is not currently waiting, the value of *Wait* is FALSE.

Timing during Waiting

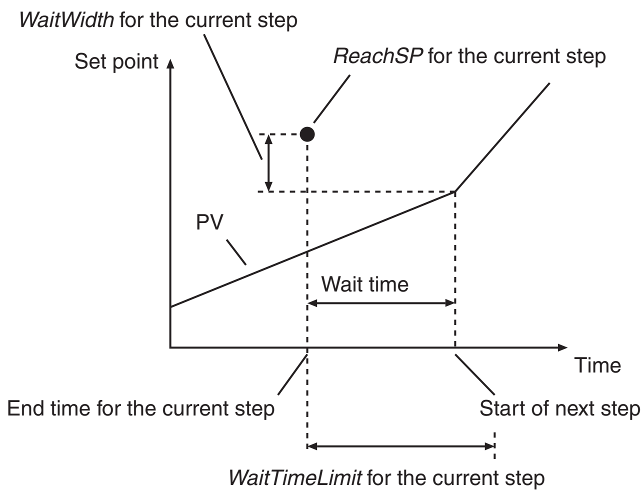
The operations of the time-related variables during waiting are described in the following table.

Name	Operation
ElapseTime	Continues timing.
ProgressTime	Stops timing and retains the value from when waiting started. Starts timing again from the retained value when waiting ends.
LeftTime	
StepProgressTime	Goes to the value of *TimeWidth* for the current step and then retains that value.
StepLeftTime	Goes to 0 and then retains that value.

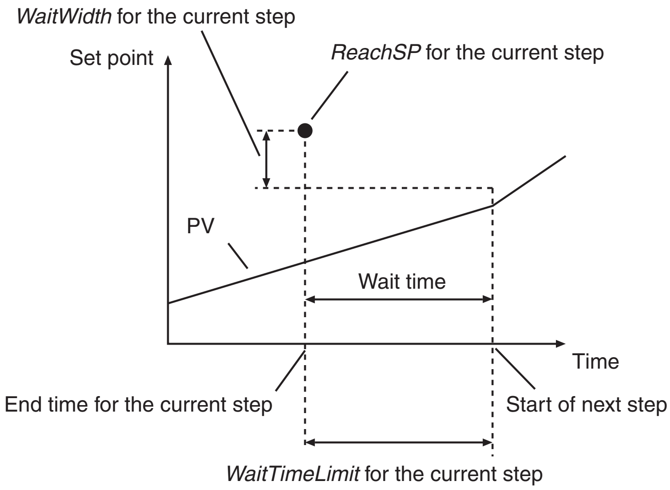
PresentSP and *PredictSP* during Waiting

During waiting, both *PresentSP* and *PredictSP* retain the value of *ReachSP*.

The following figure provides a graph of *PV* when the difference between *ReachSP* and *PV* becomes equal to or less than *WaitWidth* within the time set for *WaitTimeLimit*. The difference between *ReachSP* and *PV* still exceeds *WaitWidth* after the end time for the current step, so waiting occurs. When the difference between *ReachSP* and *PV* becomes less than or equal to *WaitWidth*, processing moves to the next step.



The following figure provides a graph of *PV* when the difference between *ReachSP* and *PV* does not become equal to or less than *WaitWidth* within the time set for *WaitTimeLimit*. Processing moves to the next step after the time that is set for *WaitTimeLimt* expires.



Holding

Processing for the current step is held unconditionally whenever the value *Hold* is TRUE. While processing is held, timing is stopped for all time-related variables.

Timing is started again for these time-related variables when the value of *Hold* changes to FALSE.

● Timing while Holding

The operations of the time-related variables while processing is held are described in the following table.

Name	Operation
ElapseTime	Stops timing and retains the value from when holding started. Starts timing again from the retained value when holding ends.
ProgressTime	
LeftTime	
StepProgressTime	
StepLeftTime	

● *PresentSP* and *PredictSP* while Holding

While processing is held, *PresentSP* retains the value from when holding started.

While processing is held, *PredictSP* has the same value as *PresentSP.*

● Holding during Waiting

If you hold processing during waiting, waiting is ended. Therefore, the value of *Wait* changes to FALSE. When holding is ended, the conditions for waiting are judged again.

Start at PV

You can start processing when the value of *PV* and the value of *PresentSP* are equal. If the value of *StartAtPV* is TRUE when *Enable* changes to TRUE, the start at PV operation is used.

Processing is performed as follows for the start at PV operation.

1. The value of *PV* is obtained.

2. A search is made from step 0 to the last step for the time when the value of *PV* first equals the value of *PresentSP*.

   If the value of *PresentSP* increases from the start of step 0, the search is made only until just before the value of *PresentSP* starts to decrease. In the same way, if the value of *PresentSP* decreases from the start of step 0, the search is made only until just before the value of *PresentSP* starts to increase.

3. Processing is started from the point that was found in the above search.

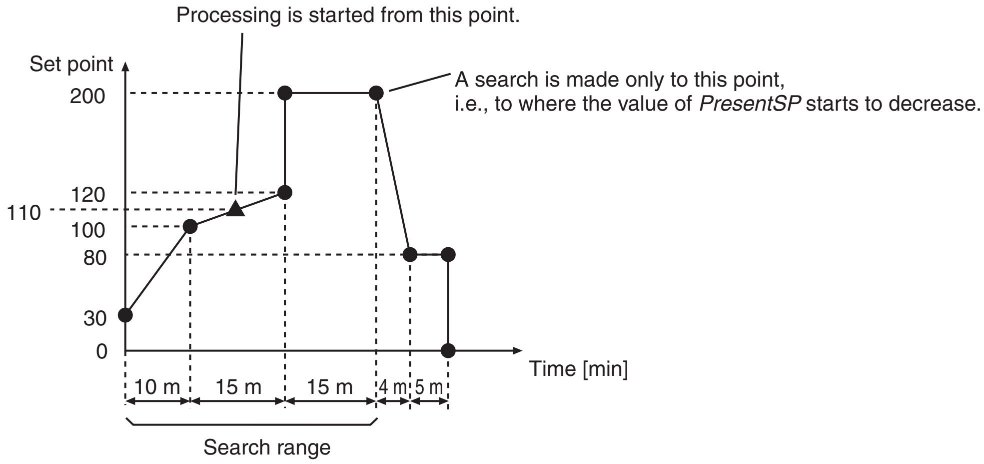
If there is no time in the search range where *PV* an *PresentSP* have the same value, processing is started from step 0.

An example of the start at PV operation is provided below. The following table gives the contents of *ProgramPattern[].*

	ProgramPattern[] element number							
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Step number	0	1	2	3	4	5	6	7
Value of *ReachSP*	30	100	120	200	200	80	80	0
Value of *TimeWidth*	T#0s	T#10m	T#15m	T#0s	T#15m	T#4m	T#5m	T#0s

In this example, the value of *PresentSP* increases from the value for step 0. Therefore, a search is made only for 40 minutes after the start of processing, i.e., the point where the value of *PresentSP* starts to decrease.

Assume that the value of *PV* at the start of instruction execution is 110. In this case, processing starts as shown in the following figure where *PresentSP* equals 110.



● Timing for Start at PV Operation

The operations of the time-related variables for the start at PV operation are described in the following table.

Name	Operation
ElapseTime	Contains 0.
ProgressTime	Gives the time from step 0 to the point that was found in the search.
LeftTime	Gives the time from the present to the end of *EndStepNo*.
StepProgressTime	Gives the time from the beginning of the current step to the point that was found in the search.
StepLeftTime	Gives the time from the present until all processing is completed for the current step.

● Changing the Value of *StartAtPV* during Instruction Execution

Any changes to the value of *StartAtPV* during execution of the instruction are ignored.

Advancing

If the value of *Advance* changes to TRUE during instruction execution, processing moves to the beginning of the next step.

● Timing for Advancing

The operations of the time-related variables when processing is advanced to the next step are described in the following table.

Name	Operation
ElapseTime	Continues timing.
ProgressTime	Gives the total of *TimeWidth* from step 0 until the current step.
LeftTime	Gives the time from the next step to the end of *EndStepNo*.
StepProgressTime	Contains 0 because processing moves to the start of the next step.
StepLeftTime	Gives the value of *TimeWidth* for the next step.

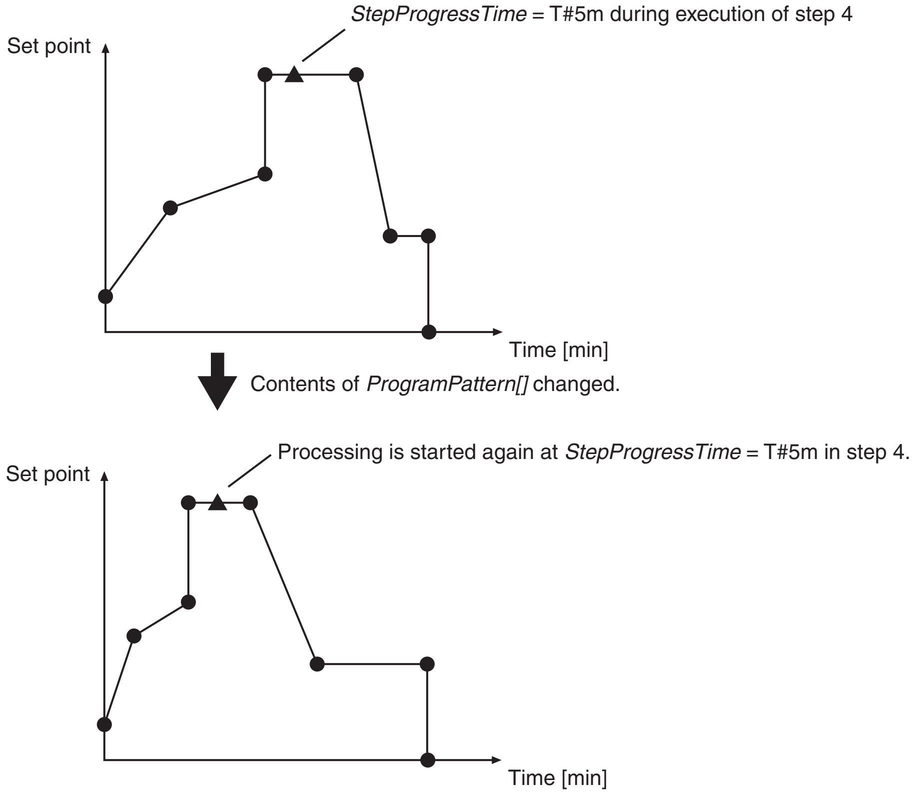
● Changing the Value of *StartStepNo* and Advancing Processing at the Same Time

If you change the values of *StartStepNo* and *Advance* to TRUE at the same time, changing the value of *StartStepNo* is given priority. Therefore, processing moves to the start of *StartStepNo*.

Changing the Program Pattern during Instruction Execution

You can change the contents of *ProgramPattern[]* during execution of the instruction. If you change the contents of *ProgramPattern[]*, the *PresentSP* is calculated again. Processing is started again from the time in *StepProgressTime* at the step that was in execution before the program pattern was changed. You can also change the contents of previous steps.

For example, assume that the contents of *ProgramPattern[]* are changed during execution of step 4. Also assume that the previous value of *StepProgressTime* was T#5m. After you change the program pattern, processing will start again at a value of T#5m for *StepProgressTime* in step 4.



If the value of *TimeWidth* for the step is smaller than the value of *StepProgressTime*, processing is started again from the start of the next step.

● Timing for Changes in the Program Pattern during Instruction Execution

The operations of the time-related variables when the program pattern is changed during instruction execution are described in the following table.

Name	Operation
ProgramTime	Gives the total of *TimeWidth* from step 0 to *EndStepNo* after the change.
ElapseTime	Continues timing.
ProgressTime	Gives the total of *StepProgressTime* and the total of *TimeWidth* from step 0 to one step before the current step after the change.
LeftTime	Gives the time from the present to the end of *EndStepNo* after the change.
StepProgressTime	Timing continues from the value before the change.
StepLeftTime	Gives the time from the present in the current step until all processing is completed for the current step after the change.

Changing the Program Pattern during Waiting

If you change the program pattern during waiting, waiting judgement is performed again for the recalculated *PresentSP*.

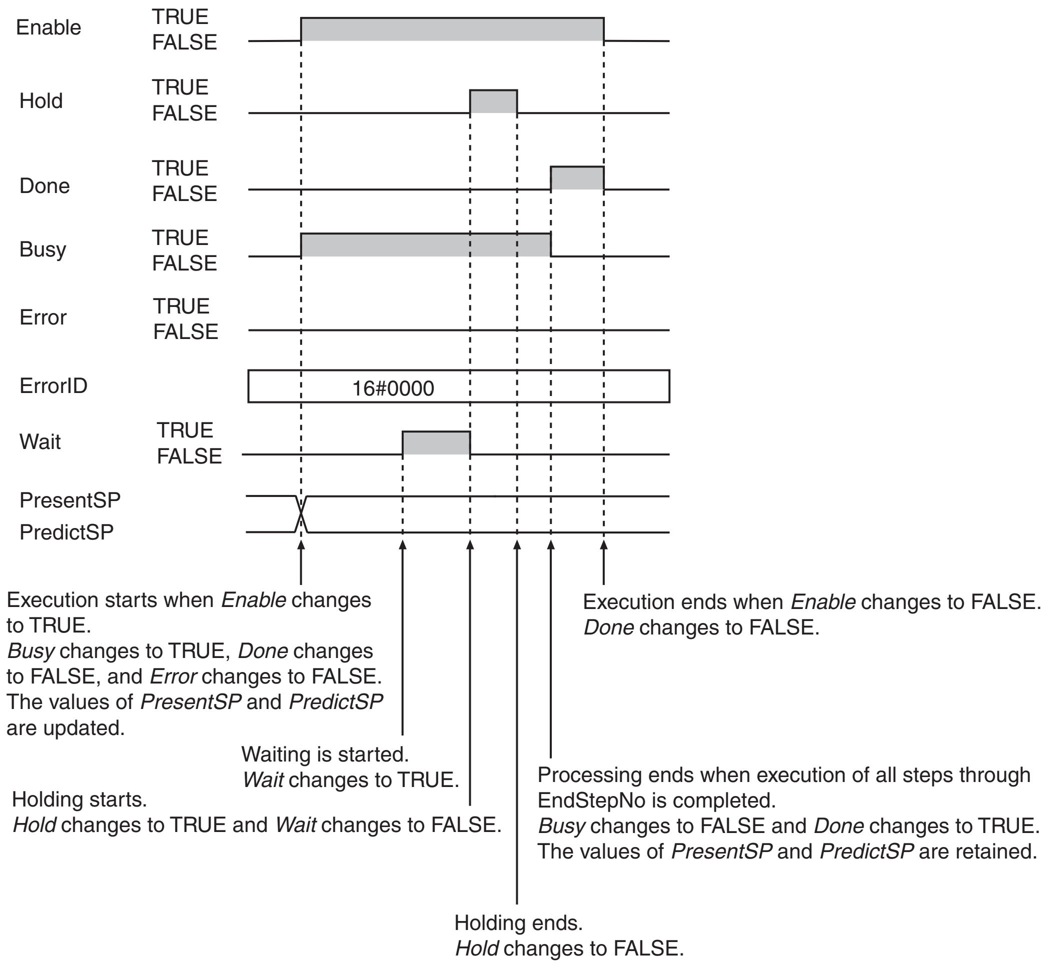
However, if the value of *StepProgressTime* is larger than the value of *WaitTimeLimit* after the change, waiting is ended immediately and processing moves to the next step.

Changing the Program Pattern during Holding

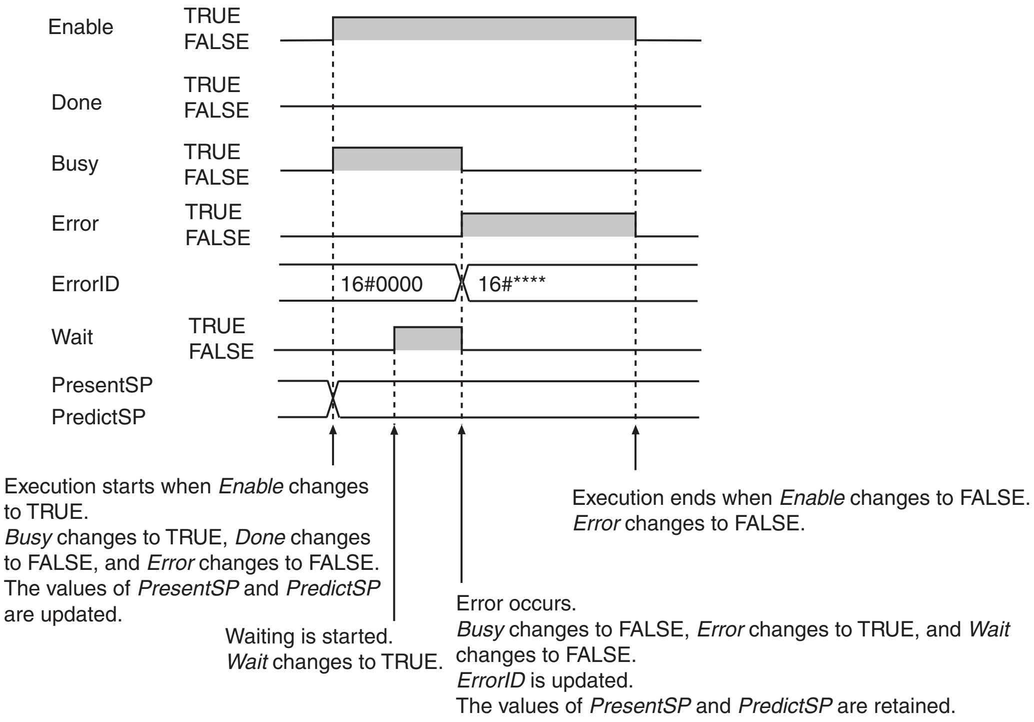
If you change the program pattern during holding, holding continues for the recalculated *PresentSP*.

Timing Charts

The following figure shows a timing chart for normal operation.



The following figure shows a timing chart for when an error occurs.



Precautions for Correct Use

- An error occurs in the following cases. *Error* will change to TRUE, and an error code is assigned to *ErrorID.*

Error	Value of *ErrorID*
The value of *IntegrationTime, Alpha, StartStepNo*, or *EndStepNo* is outside of the valid range.	16#0400
The final element number in the *ProgramPattern[]* array exceeded 99.	16#0416

Version Information

A CPU Unit with unit version 1.06 or later and Sysmac Studio version 1.07 or higher are required to use this instruction.

Sample Programming

This sample performs temperature control with the optimum PID parameters for each step in the AC_-StepProgram instruction.

Processing

This sample performs the following two processes.

- It calculates the optimum PID parameters for each step.
- It controls temperature according to the program pattern.

Both of these processes are described below.

● Calculating Optimum PID Parameters for Each Step

Before temperature is controlled according to the program pattern, the optimum PID parameters for each step must be calculated. Autotuning with the PIDAT instruction is used to calculate the PID parameters.

The calculated PID parameters are stored in the *PIDbank[]* array of structures with the step numbers used as the array subscripts. The members of the elements of *PIDbank[]* give the proportional bands, integration times, and derivative times.

The processing procedure is as follows:

1 The user changes the value of *ACSP_Enable* to the AC_StepProgram instruction to TRUE.

The AC_StepProgram instruction is executed and the value of present step number *StepNo* changes to 0.

2 The user changes the value of execution condition *Run* to the PIDAT instruction to TRUE.

The PIDAT instruction is executed.

3 The user changes the value of autotuning execution condition *StartAT* to TRUE.

The value of *Hold* to the AC_StepProgram instruction changes to TRUE and holding is performed. Autotuning for the PIDAT instruction is executed and the optimum PID parameters are calculated for step 0.

4 Autotuning is completed.

The value of autotuning normal completion *ATDone* from the PIDAT instruction changes to TRUE.

The calculated PID parameters are stored in *PIDbank[0].*

5 The user changes the value of *Hold* to the AC_StepProgram instruction to FALSE.

Holding for the AC_StepProgram instruction is canceled.

After a while, processing moves to the next step and the value of *StepNo* changes to 1.

6 The user repeats steps 3 to 5 for each step number.

The optimum PID parameters for all steps are stored in *PIDbank[].*

● Controlling Temperature According to the Program Pattern

The optimum PID parameters for each step are used to control temperature according to the program pattern.

The processing procedure is as follows:

1. The user changes the value of *ACSP_Enable* to the AC_StepProgram instruction to TRUE.

   The AC_StepProgram instruction is executed and the value of step number *StepNo* changes to 0.

2. The user changes the value of execution condition *Run* to the PIDAT instruction to TRUE.

   The PIDAT instruction is executed.

3. For each task period, manipulated value *MV* from the PIDAT instruction is output.

4. The TimeProportionalOut instruction performs time-proportional output according to the value of *MV*.

5. After a while, processing moves to the next step.

6. Steps 3 to 5 are repeated through the end step.

Setup with the Sysmac Studio

To use the sample programming, you must use the Sysmac Studio to set the network configuration, I/O map, and data type definitions.

● Network Settings

The configuration of the network is given in the following table. A Slave Terminal with the following configuration is connected at EtherCAT node address 1. The device names that are given in the following table are used.

Unit number	Model number	Unit	Device name
0	NX-ECC201	EtherCAT Coupler Unit	E001
1	NX-TS2101	Temperature Input Unit	N1
2	NX-OD3121	Digital Output Unit	N2

● I/O Map

The following I/O map settings are used.

Position	Port	Description	R/W	Data type	Variable	Variable type
Unit1	Ch1 Measured Value REAL*1	Channel measured value (REAL)	R	REAL	N1_Ch1_Measured_Value_REAL	Global variable
Unit1	Ch2 Measured Value REAL *2	Channel measured value (REAL)	R	REAL	N1_Ch2_Measured_Value_REAL	Global variable
Unit2	Output Bit 00	Output bit 00	W	BOOL	N2_Output_Bit_00	Global variable
Unit2	Output Bit 01	Output bit 01	W	BOOL	N2_Output_Bit_01	Global variable
Unit2	Output Bit 02	Output bit 02	W	BOOL	N2_Output_Bit_02	Global variable
Unit2	Output Bit 03	Output bit 03	W	BOOL	N2_Output_Bit_03	Global variable

*1 You must add 0x6003:01 (Ch1 Measured Value REAL) to the I/O entries for the NX-TS2101 Temperature Input Unit.

*2 You must add 0x6003:02 (Ch2 Measured Value REAL) to the I/O entries for the NX-TS2101 Temperature Input Unit.

● Data Type Definitions

The structure *sPID_BANK* is defined as shown in the following table.

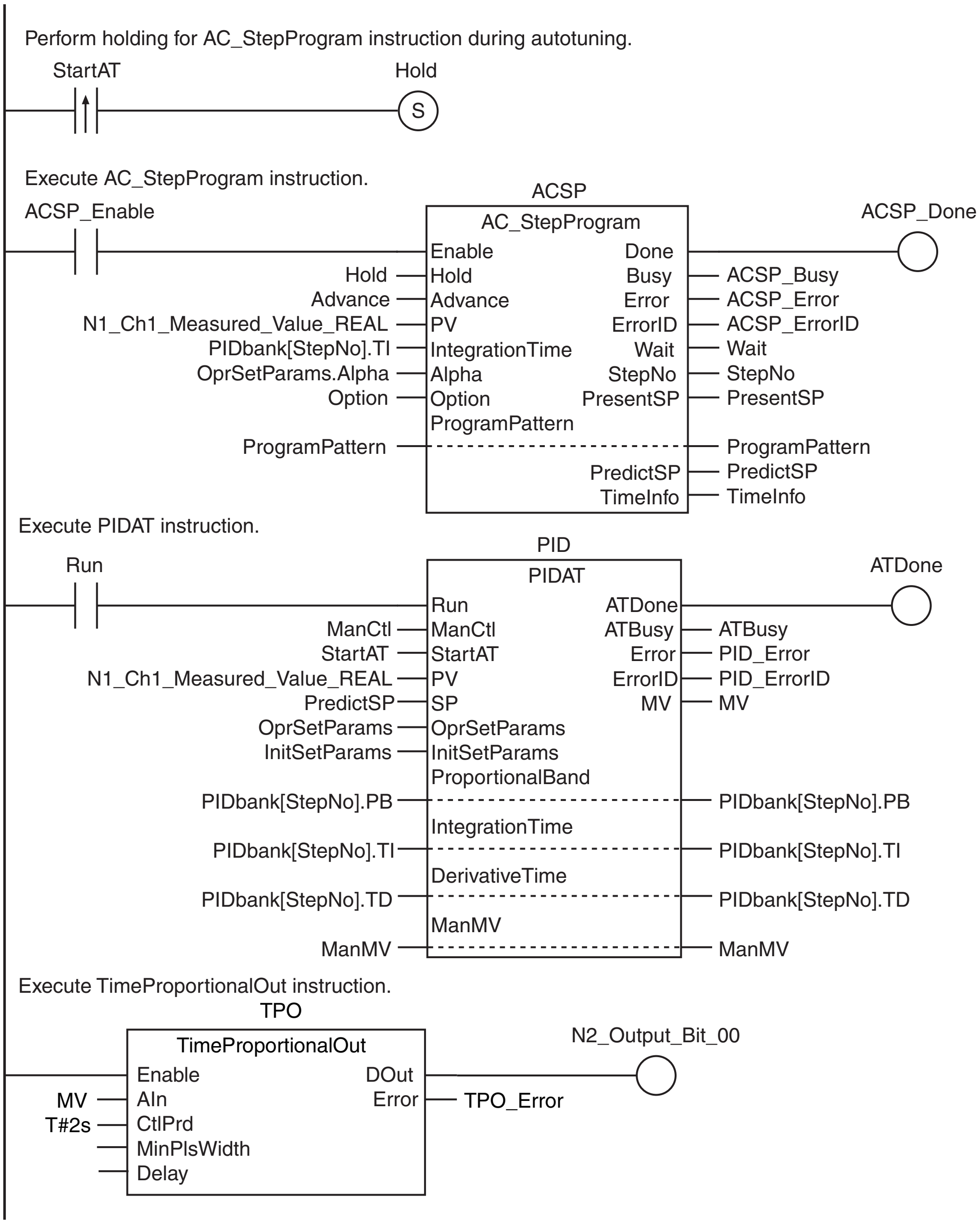
Structure	Name	Data type	Comment
▼	sPID_BANK	STRUCT	PID parameter structure
	PB	REAL	Proportional band
	TI	TIME	Integration time
	TD	TIME	Derivative time

LD

Internal Variables	Variable	Data type	Initial value	Comment
	ACSP_Enable	BOOL	FALSE	Enable for AC_StepProgram
	Hold	BOOL	FALSE	Hold
	Advance	BOOL	FALSE	Advance
	Option	_sAC_STEP_OPTION	(StartAtPV:=FALSE, StartStepNo:=0, EndStepNo:=7, Reserved:=[32(16#0)])	Option
	ProgramPattern	ARRAY[0..7] OF _sAC_STEP_DATA	[(ReachSP:=30.0, TimeWidth:=T#0s, WaitWidth:=3.0, WaitTimeLimit:=T#1m), (ReachSP:=100.0, TimeWidth:=T#10m, WaitWidth:=3.0, WaitTimeLimit:=T#1m), (ReachSP:=120.0, TimeWidth:=T#15m, WaitWidth:=3.0, WaitTimeLimit:=T#1m), (ReachSP:=150.0, TimeWidth:=T#0s, WaitWidth:=3.0, WaitTimeLimit:=T#1m), (ReachSP:=150.0, TimeWidth:=T#15m, WaitWidth:=3.0, WaitTimeLimit:=T#1m), (ReachSP:=80.0, TimeWidth:=T#4m, WaitWidth:=3.0, WaitTimeLimit:=T#1m), (ReachSP:=80.0, TimeWidth:=T#5m, WaitWidth:=3.0, WaitTimeLimit:=T#1m), (ReachSP:=10.0, TimeWidth:=T#0s, WaitWidth:=3.0, WaitTimeLimit:=T#1m)]	Program pattern
	ACSP_Busy	BOOL	FALSE	Execution of AC_StepProgram in progress
	ACSP_Error	BOOL	FALSE	AC_StepProgram error
	ACSP_ErrorID	WORD	WORD#16#0	AC_StepProgram error code
	Wait	BOOL	FALSE	Waiting
	StepNo	USINT	0	Present step number
	PresentSP	REAL	0.0	Present set point
	PredictSP	REAL	0.0	Predicted set point

Internal Variables	Variable	Data type	Initial value	Comment
	TimeInfo	_sAC_STEP_TIME	(ProgramTime:=T#0s, ElapseTime:=T#0s, ProgressTime:=T#0s, LeftTime:=T#0s, StepProgressTime:=T#0s, StepLeftTime:=T#0s)	Clock information
	ACSP_Done	BOOL	FALSE	AC_StepProgram completion
	Run	BOOL	FALSE	PIDAT instruction execution condition
	ManCtl	BOOL	FALSE	Manual/auto control
	StartAT	BOOL	FALSE	Autotuning execution condition
	OprSetParams	_sOPR_SET _PARAMS	(MVLowLmt:=0.0, MVUpLmt:=100.0, ManResetVal:=0.0, MVTrackSw:=FALSE, MVTrackVal:=0.0, StopMV:=0.0, ErrorMV:=0.0, Alpha:=0.65, ATCalcGain:=1.0, ATHystrs:=0.2)	Operation setting parameters
	InitSetParams	_sINIT_SET _PARAMS	(SampTime:=T#250ms, RngLowLmt:=-200.0, RngUpLmt:=1300.0, DirOpr:=FALSE)	Initial setting parameters
	ManMV	REAL	0.0	Manual manipulated variable
	ATBusy	BOOL	FALSE	Autotuning busy
	PID_ErrorID	WORD	WORD#16#0	PIDAT error code
	PID_Error	BOOL	FALSE	PIDAT error
	MV	REAL	0.0	Manipulated variable
	ATDone	BOOL	FALSE	Autotuning normal completion
	TPO_Error	BOOL	FALSE	TimeProportionalOut error
	PIDbank	ARRAY[0..7] OF sPID_BANK	[8((PB:=10, TI:=T#233s, TD:=T#60s))]	Storage array for optimum PID parameters
	ACSP	AC_StepProgram		
	PID	PIDAT		
	TPO	TimeProportionalOut		

External Variables	Variable	Data type	Constant	Comment
	N1_Ch1_Measured_Value_REAL	REAL	---	Channel measured value (REAL)
	N2_Output_Bit_00	BOOL	---	Output bit


Perform holding for AC_StepProgram instruction during autotuning.
StartAT
Hold
S
Execute AC_StepProgram instruction.
ACSP
ACSP_Enable
AC_StepProgram
ACSP_Done
Enable
Done
Hold
Hold
Busy
ACSP_Busy
Advance
Advance
Error
ACSP_Error
N1_Ch1_Measured_Value_REAL
PV
ErrorID
ACSP_ErrorID
PIDbank[StepNo].TI
IntegrationTime
Wait
Wait
OprSetParams.Alpha
Alpha
StepNo
StepNo
Option
Option
PresentSP
PresentSP
ProgramPattern
ProgramPattern
ProgramPattern
PredictSP
PredictSP
TimeInfo
TimeInfo
Execute PIDAT instruction.
PID
Run
PIDAT
ATDone
Run
ATDone
ManCtl
ManCtl
ATBusy
ATBusy
StartAT
StartAT
Error
PID_Error
N1_Ch1_Measured_Value_REAL
PV
ErrorID
PID_ErrorID
PredictSP
SP
MV
MV
OprSetParams
OprSetParams
InitSetParams
InitSetParams
ProportionalBand
PIDbank[StepNo].PB
PIDbank[StepNo].PB
IntegrationTime
PIDbank[StepNo].TI
PIDbank[StepNo].TI
DerivativeTime
PIDbank[StepNo].TD
PIDbank[StepNo].TD
ManMV
ManMV
ManMV
Execute TimeProportionalOut instruction.
TPO
TimeProportionalOut
N2_Output_Bit_00
Enable
DOut
MV
AIn
Error
TPO_Error
T#2s
CtlPrd
MinPlsWidth
Delay

ST

Internal Variables	Variable	Data type	Initial value	Comment
	ACSP_Enable	BOOL	FALSE	Enable for AC_StepProgram
	Hold	BOOL	FALSE	Hold
	Advance	BOOL	FALSE	Advance
	Option	_sAC_STEP_OPTION	(StartAtPV:=FALSE, StartStepNo:=0, EndStepNo:=7, Reserved:=[32(16#0)])	Option
	ProgramPattern	ARRAY[0..7] OF _sAC_STEP_DATA	[(ReachSP:=30.0, TimeWidth:=T#0s, WaitWidth:=3.0, WaitTimeLimit:=T#1m), (ReachSP:=100.0, TimeWidth:=T#10m, WaitWidth:=3.0, WaitTimeLimit:=T#1m), (ReachSP:=120.0, TimeWidth:=T#15m, WaitWidth:=3.0, WaitTimeLimit:=T#1m), (ReachSP:=150.0, TimeWidth:=T#0s, WaitWidth:=3.0, WaitTimeLimit:=T#1m), (ReachSP:=150.0, TimeWidth:=T#15m, WaitWidth:=3.0, WaitTimeLimit:=T#1m), (ReachSP:=80.0, TimeWidth:=T#4m, WaitWidth:=3.0, WaitTimeLimit:=T#1m), (ReachSP:=80.0, TimeWidth:=T#5m, WaitWidth:=3.0, WaitTimeLimit:=T#1m), (ReachSP:=10.0, TimeWidth:=T#0s, WaitWidth:=3.0, WaitTimeLimit:=T#1m)]	Program pattern
	ACSP_Busy	BOOL	FALSE	Execution of AC_StepProgram in progress
	ACSP_Error	BOOL	FALSE	AC_StepProgram error
	ACSP_ErrorID	WORD	WORD#16#0	AC_StepProgram error code
	Wait	BOOL	FALSE	Waiting
	StepNo	USINT	0	Present step number
	PresentSP	REAL	0.0	Present set point
	PredictSP	REAL	0.0	Predicted set point

Internal Variables	Variable	Data type	Initial value	Comment
	TimeInfo	_sAC_STEP_TIME	(ProgramTime:=T#0s, ElapseTime:=T#0s, ProgressTime:=T#0s, LeftTime:=T#0s, StepProgressTime:=T#0s, StepLeftTime:=T#0s)	Clock information
	ACSP_Done	BOOL	FALSE	AC_StepProgram completion
	Run	BOOL	FALSE	PIDAT instruction execution condition
	ManCtl	BOOL	FALSE	Manual/auto control
	StartAT	BOOL	FALSE	Autotuning execution condition
	PreStartAT	BOOL	TRUE	Autotuning execution condition for previous task period
	OprSetParams	_sOPR_SET _PARAMS	(MVLowLmt:=0.0, MVUpLmt:=100.0, ManResetVal:=0.0, MVTrackSw:=FALSE, MVTrackVal:=0.0, StopMV:=0.0, ErrorMV:=0.0, Alpha:=0.65, ATCalcGain:=1.0, ATHystrs:=0.2)	Operation setting parameters
	InitSetParams	_sINIT_SET _PARAMS	(SampTime:=T#250ms, RngLowLmt:=-200.0, RngUpLmt:=1300.0, DirOpr:=FALSE)	Initial setting parameters
	ManMV	REAL	0.0	Manual manipulated variable
	ATBusy	BOOL	FALSE	Autotuning busy
	PID_ErrorID	WORD	WORD#16#0	PIDAT error code
	PID_Error	BOOL	FALSE	PIDAT error
	MV	REAL	0.0	Manipulated variable
	ATDone	BOOL	FALSE	Autotuning normal completion
	TPO_Error	BOOL	FALSE	TimeProportionalOut error
	PIDbank	ARRAY[0..7] OF sPID_BANK	[8((PB:=10, TI:=T#233s, TD:=T#60s))]	Storage array for optimum PID parameters
	TPO_Enable	BOOL	FALSE	Enable for TimeProportionalOut
	MinPlsWidth	REAL	0.0	Minimum pulse width
	Delay	REAL	0.0	Delay
	ACSP	AC_StepProgram		
	PID	PIDAT		
	TPO	TimeProportionalOut		

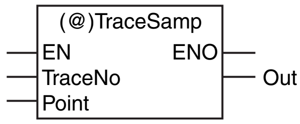
External Variables	Variable	Data type	Constant	Comment
	N1_Ch1_Measured_Value_REAL	REAL	---	Channel measured value (REAL)
	N2_Output_Bit_00	BOOL	---	Output bit

```
TPO_Enable := TRUE;

// Perform holding for AC_StepProgram instruction during autotuning.
IF StartAT AND PreStartAT=FALSE THEN
   Hold := TRUE;
END_IF;
PreStartAT := StartAT;

// Execute AC_StepProgram instruction.
IF ACSP_Enable THEN
   ACSP(Enable         :=ACSP_Enable,
        Hold           :=Hold,
        Advance        :=Advance,
        PV             :=N1_Ch1_Measured_Value_REAL,
        IntegrationTime:=PIDbank[StepNo].TI,
        Alpha          :=OprSetParams.Alpha,
        Option         :=Option,
        ProgramPattern :=ProgramPattern,
        Done           =>ACSP_Done,
        Busy           =>ACSP_Busy,
        Error          =>ACSP_Error,
        ErrorID        =>ACSP_ErrorID,
        Wait           =>Wait,
        StepNo         =>StepNo,
        PresentSP      =>PresentSP,
        PredictSP      =>PredictSP,
        TimeInfo       =>TimeInfo);
END_IF;

// Execute PIDAT instruction.
IF Run THEN
   PID(Run              :=Run,
       ManCtl           :=ManCtl,
       StartAT          :=StartAT,
       PV               :=N1_Ch1_Measured_Value_REAL,
       SP               :=PredictSP,
       OprSetParams     :=OprSetParams,
       InitSetParams    :=InitSetParams,
       ProportionalBand:=PIDbank[StepNo].PB,
       IntegrationTime  :=PIDbank[StepNo].TI,
       DerivativeTime   :=PIDbank[StepNo].TD,
       ManMV            :=ManMV,
       ATDone           =>ATDone,
       ATBusy           =>ATBusy,
       Error            =>PID_Error,
       ErrorID          =>PID_ErrorID,
       MV=>MV);
END_IF;

// Execute TimeProportionalOut instruction.
TPO(Enable     :=TPO_Enable,
    AIn        :=MV,
    CtlPrd     :=T#2s,
    MinPlsWidth:=MinPlsWidth,
    Delay      :=Delay,
    DOut       =>N2_Output_Bit_00,
    Error      =>TPO_Error);
```

System Control Instructions

TraceSamp

The TraceSamp instruction performs sampling for a data trace.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TraceSamp	Data Trace Sampling	FUN		TraceSamp(TraceNo, Point);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
TraceNo	Trace number	Input	Trace number	*1	---	0
Point	Sampling point number	Input	Sampling point number	Depends on data type.	---	0
Out	Return value	Output	Always TRUE	TRUE only	---	---

*1 The range is 0 to 3 for an NX701 or NJ501 CPU Unit, and for an NY-series Controller.
The range is 0 to 1 for an NX1P2, NJ301 or NJ101 CPU Unit.

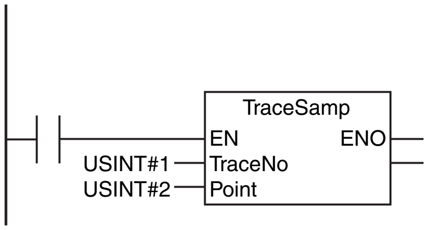
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
TraceNo						OK														
Point						OK														
Out	OK																			

Function

The TraceSamp instruction performs sampling for a data trace. The sampling settings are specified from the Sysmac Studio. The present values for all variables that are set to be sampled are read and stored with trace number *TraceNo* and sampling point number *Point* in trace memory. This instruction is executed only during execution of data tracing and only when the sampling timing is set to sampling instructions from the Sysmac Studio.

The following figure shows a programming example. Trace number 1 and sampling point number 2 are attached, and the present values of all variables to be sampled are stored in trace memory.

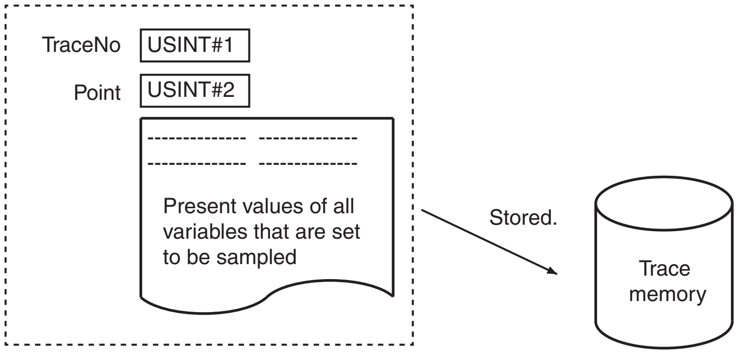
LD



ST

```
TraceSamp(USINT#1, USINT#2);
```

The present values for all variables that are set to be sampled are read and stored with trace number *TraceNo* and sampling point number *Point* in trace memory.



Related System-defined Variables

Name	Meaning	Data type	Description
*1	Trace Information	*2	Trace information*3

*1 NX701 or NJ501 CPU Unit, and NY-series Controller: The variable name is *_PLC_TraceSta[0..3]*.
NX1P2, NJ301 or NJ101 CPU Unit: The variable name is *_PLC_*TraceSta*[0..1]*.

*2 _sTRACE_STA[]

*3 Refer to the *NJ/NX-series CPU Unit Software User's Manual* (Cat. No. W501) or *NY-series Industrial Panel PC / Industrial Box PC Software User's Manual* (Cat. No. W558) for details.

Additional Information

- Refer to the *NJ/NX-series CPU Unit Software User's Manual* (Cat. No. W501) or *NY-series Industrial Panel PC / Industrial Box PC Software User's Manual* (Cat. No. W558) for details on data tracing.
- Tracing is used to sample the values of specified variables under specified conditions. The conditions are specified from the Sysmac Studio.
- This instruction can be located in more than one place in the user program. Programming can be written to sample according to specific conditions.
- *Point* can be suitably set so that you can see which sampled values on the Data Trace Window in the Sysmac Studio were returned by which TraceSamp instruction. *Point* will default to 0 if it is omitted.

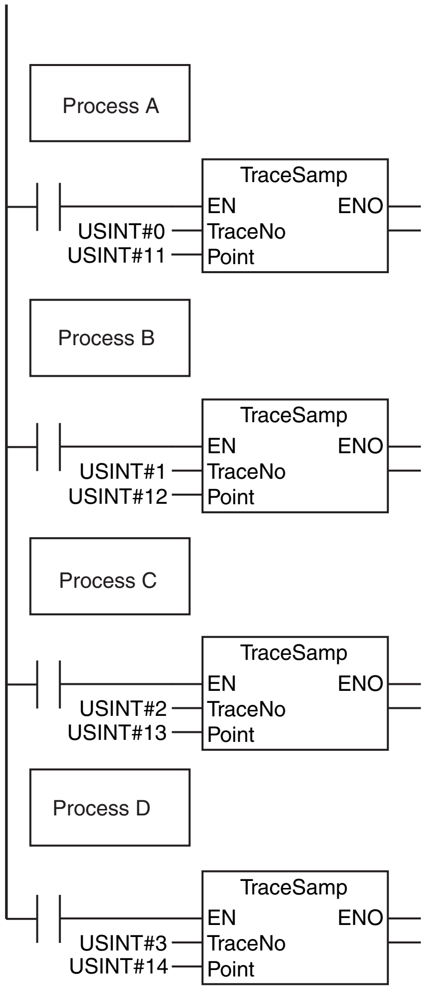
Precautions for Correct Use

- Return value *Out* is not used when the instruction is used in ST.
- In the following cases, nothing is done and the instruction ends normally.
  - Data tracing is stopped.
  - The sampling timing is not set to sampling instructions in the trace settings.
  - The value of *TraceNo* is not the trace number set from the Sysmac Studio.
- An error occurs in the following case. *ENO* will be FALSE.
  - The value of *TraceNo* is outside of the valid range.

Sample Programming

Here, sampling is performed at the end of each process A to D. The values of the variables are stored at each point.

LD



ST

```
Process A
TraceSamp(USINT#0, USINT#11);
Process B
TraceSamp(USINT#1, USINT#12);
Process C
TraceSamp(USINT#2, USINT#13);
Process D
TraceSamp(USINT#3, USINT#14);
```

TraceTrig

The TraceTrig instruction generates a trigger for data tracing.

Instruction	Name	FB/FUN	Graphic expression	ST expression
TraceTrig	Data Trace Trigger	FUN	(@)TraceTrig EN ENO TraceNo Out	TraceTrig(TraceNo);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
TraceNo	Trace number	Input	Trace number	*1	---	0
Out	Return value	Output	Always TRUE	TRUE only	---	---

*1 The range is 0 to 3 for an NX701 or NJ501 CPU Unit, and for an NY-series Controller.
The range is 0 to 1 for an NX1P2, NJ301 or NJ101 CPU Unit.

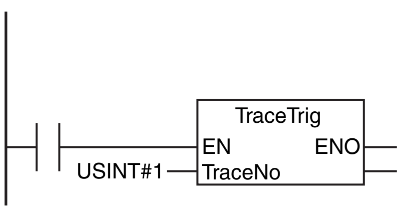
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
TraceNo						OK														
Out	OK																			

Function

The TraceTrig instruction generates a trigger for data tracing. It does not matter whether the trigger conditions that were set from the Sysmac Studio have been met. Sampling starts if data tracing is in progress for trace number *TraceNo* when the instruction is executed.

The following figure shows a programming example. Here, a data trace trigger is generated for trace number 1.

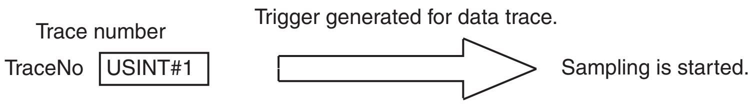
LD



ST

```
TraceTrig(USINT#1);
```

Here, a data trace trigger is generated for trace number *TraceNo*.



Related System-defined Variables

Name	Meaning	Data type	Description
*1	Trace Information	*2	Trace information*3

*1 NX701 or NJ501 CPU Unit, and NY-series Controller: The variable name is *_PLC_TraceSta[0..3]*.
NX1P2, NJ301 or NJ101 CPU Unit: The variable name is *_PLC_*TraceSta*[0..1]*.

*2 _sTRACE_STA[]

*3 Refer to the *NJ/NX-series CPU Unit Software User's Manual* (Cat. No. W501) or *NY-series Industrial Panel PC / Industrial Box PC Software User's Manual* (Cat. No. W558) for details.

Additional Information

- Refer to the *NJ/NX-series CPU Unit Software User's Manual* (Cat. No. W501) or *NY-series Industrial Panel PC / Industrial Box PC Software User's Manual* (Cat. No. W558) for details on data tracing.
- This instruction can be located in more than one place in the user program. Programming can be written to generate a trigger according to specific conditions.
- Programming can be written to generate triggers in ways that are not possible for normal trigger conditions settings, such as programming to generate a trigger based on a comparison of two variables.

Precautions for Correct Use

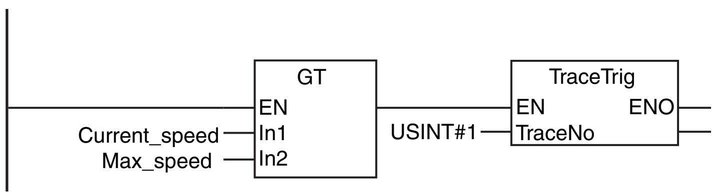
- Return value *Out* is not used when the instruction is used in ST.
- In the following cases, nothing is done and the instruction ends normally.
  - Data tracing is stopped.
  - The trigger condition has already been met.
  - The value of *TraceNo* is not the trace number set from the Sysmac Studio.
  - A continuous trace is specified as the trace type for the trace number that is specified with *TraceNo*.
- An error occurs in the following case. *ENO* will be FALSE.
  - The value of *TraceNo* is outside of the valid range.

Sample Programming

Here, a data trace trigger is generated to store the values of variables when the current speed exceeds the maximum speed. The TraceTrig instruction is executed when the value of *Current_speed* exceeds the value of *Max_speed*.

LD

Variable	Data type	Initial value	Comment
Current_speed	INT	0	Current speed
Max_speed	INT	20	Maximum speed



ST

Variable	Data type	Initial value	Comment
Current_speed	INT	0	Current speed
Max_speed	INT	20	Maximum speed

```
IF (Current_speed > Max_speed) THEN
    TraceTrig(USINT#1);
END_IF;
```

GetTraceStatus

The GetTraceStatus instruction reads the execution status of a data trace.

Instruction	Name	FB/FUN	Graphic expression	ST expression
GetTraceStatus	Read Data Trace Status	FUN	(@)GetTraceStatus EN ENO TraceNo Out IsStart IsComplete ParamErr IsTrigger	GetTraceStatus(TraceNo, IsStart, IsComplete, ParamErr, IsTrigger);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
TraceNo	Trace number	Input	Trace number	*1	---	0
Out	Return value	Output	Always TRUE	TRUE only	---	---
IsStart	Executing flag	Output	TRUE: Data trace in progress. FALSE: Data trace not in progress.	Depends on data type.	---	---
IsComplete	Completed flag	Output	TRUE: Data trace was completed. FALSE: Data trace in progress or not executed.	Depends on data type.	---	---
ParamErr	Parameter error flag	Output	TRUE: Data trace setting error. FALSE: No data trace setting error.	Depends on data type.	---	---
IsTrigger	Trigger flag	Output	TRUE: Data trace trigger condition met. FALSE: Data trace trigger condition not met.	Depends on data type.	---	---

*1 The range is 0 to 3 for an NX701 or NJ501 CPU Unit, and for an NY-series Controller.
The range is 0 to 1 for an NX1P2, NJ301 or NJ101 CPU Unit.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
TraceNo						OK														
Out	OK																			
IsStart	OK																			

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
IsComplete	OK																			
ParamErr	OK																			
IsTrigger	OK																			

Function

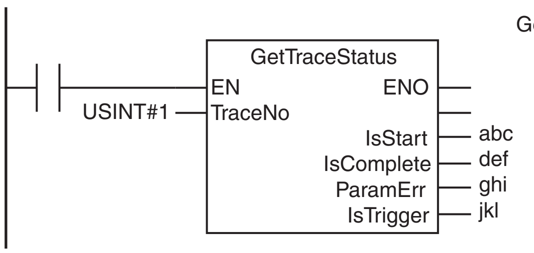
The GetTraceStatus instruction reads the execution status of the data trace that is specified with trace number *TraceNo*. The status that is read is output to execution flag *IsStart*, completed flag *IsComplete*, parameter error flag *ParamErr*, and trigger flag *IsTrigger*.

The value of *ParamErr* changes to TRUE when one of the following errors is found in the trace settings.

- A variable that is specified in the trigger or sampling settings does not exist.
- Sampling is set to be performed on a specified task period, but the specified task does not exist.

The following figure shows a programming example. The GetTraceStatus instruction reads the execution status of the data trace with trace number 1.

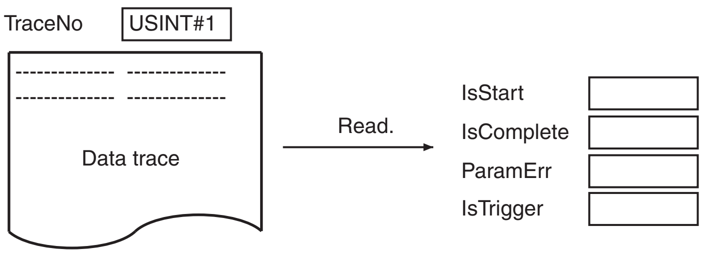
LD



ST

```
GetTraceStatus(USINT#1, abc, def, ghi, jkl);
```

The GetTraceStatus instruction reads the execution status of the data trace that is specified with trace number *TraceNo*.



Related System-defined Variables

Name	Meaning	Data type	Description
*1	Trace Information	*2	Trace information*3

*1 NX701 or NJ501 CPU Unit, and NY-series Controller: The variable name is *_PLC_TraceSta[0..3]*.
NX1P2, NJ301 or NJ101 CPU Unit: The variable name is *_PLC_*TraceSta*[0..1]*.

*2 _sTRACE_STA[]

*3 Refer to the *NJ/NX-series CPU Unit Software User's Manual* (Cat. No. W501) or *NY-series Industrial Panel PC / Industrial Box PC Software User's Manual* (Cat. No. W558) for details.

Additional Information

Refer to the *NJ/NX-series CPU Unit Software User's Manual* (Cat. No. W501) or *NY-series Industrial Panel PC / Industrial Box PC Software User's Manual* (Cat. No. W558) for details on data tracing.

Precautions for Correct Use

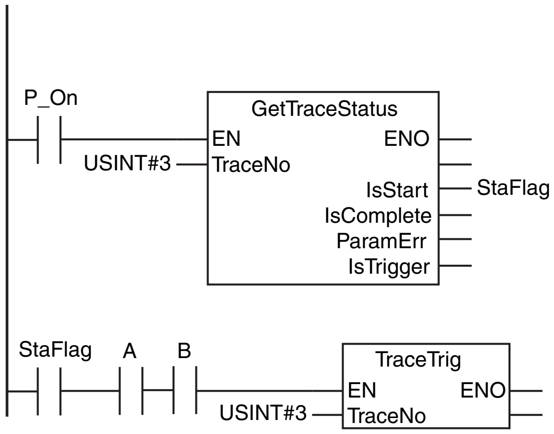
- Return value *Out* is not used when the instruction is used in ST.
- This instruction reads the contents of the *_PLC_TraceSta[]* system-defined variable. You cannot access this variable directly. Always use this instruction to read the contents of the variable.
- An error occurs in the following case. *ENO* will be FALSE.
  - The value of *TraceNo* is outside of the valid range.

Sample Programming

In this sample, the GetTraceStatus instruction reads the execution status of the data trace with trace number 3. If the data trace is in progress, the TraceTrig instruction is executed to trigger data tracing.

LD

Variable	Data type	Initial value	Comment
StaFlag	BOOL	FALSE	Trace execution status
A	BOOL	FALSE	
B	BOOL	FALSE	



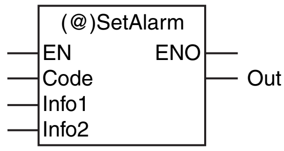
ST

Variable	Data type	Initial value	Comment
StaFlag	BOOL	FALSE	Trace execution status
A	BOOL	FALSE	
B	BOOL	FALSE	

```
GetTraceStatus(TraceNo:=USINT#3, IsStart=>StaFlag);

IF ( (StaFlag=TRUE) AND (A=TRUE) AND (B=TRUE) ) THEN
    TraceTrig(TraceNo:=USINT#3);
END_IF;
```

SetAlarm

The SetAlarm instruction creates a user-defined error.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SetAlarm	Create User-defined Error	FUN		SetAlarm(Code, Info1, Info2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Code	Event code	Input	Event code of user-defined error to generate	1 to 40000	---	1
Info1	Attached information 1		Values recorded in event log when the user-defined error is generated	Depends on data type.		*
Info2	Attached information 2					
Out	Return value	Output	Always TRUE	TRUE only	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

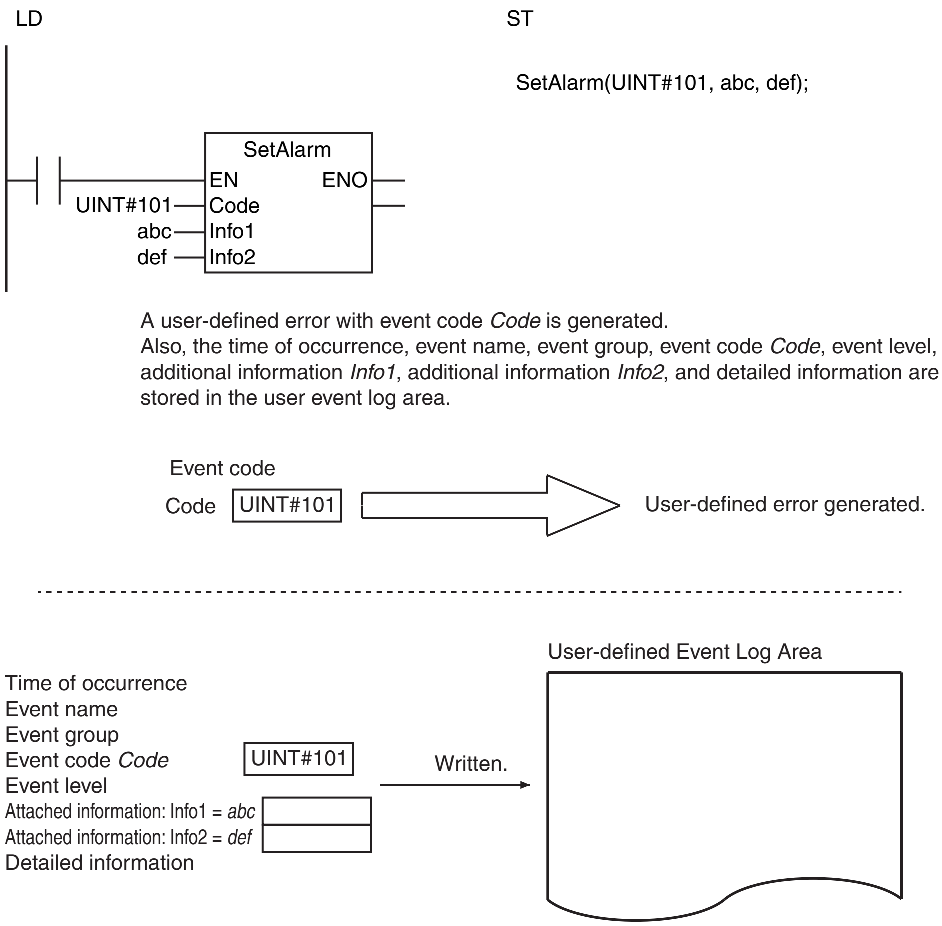
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Code							OK													
Info1	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	
Info2	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	
Out	OK																			

Function

The SetAlarm instruction generates the user-defined error that corresponds to event code *Code*. Event codes are defined in the event setting table on the Sysmac Studio. The time of occurrence, event name, event group, event code *Code*, event level, additional information *Info1*, additional information *Info2*, and detailed information are stored in the user event log area that corresponds to the level of the event code. The value for the time of occurrence is automatically obtained. The event name, event group, and detailed information that are set from the Sysmac Studio are recorded. The event level that corresponds to the event code is recorded. The event levels are given below. The smaller the event code is, the higher the event level is.

Event code	Classification: User fault level
1 to 5000	1
5001 to 10000	2
10001 to 15000	3
15001 to 20000	4
20001 to 25000	5
25001 to 30000	6
30001 to 35000	7
35001 to 40000	8

The following figure shows a programming example. A user-defined error with event code 101 is generated. The values of variables *abc* and *def* are stored as attached information.



Related System-defined Variables

Name	Meaning	Data type	Description
_AlarmFlag	Error Status of User-defined Errors	WORD	These flags indicate when user-defined errors are detected. Bit 0 to bit 7 indicate the status of user-defined error levels 1 to 8.[*1]

*1 Refer to the *NJ/NX-series CPU Unit Software User's Manual* (Cat. No. W501) or *NY-series Industrial Panel PC / Industrial Box PC Software User's Manual* (Cat. No. W558) for details.

Additional Information

You can specify either global variables or local variables for *Info1* and *Info2*.

Precautions for Correct Use

- Up to 32 user-defined errors can be generated in each of the eight event levels (for up to 256 user-defined errors total).
- If a user-defined error for the same event code already exists, the new error is not recorded in the event log.
- Always use variables for the input parameters that pass *Info1* and *Info2*. If you use a constant, a building error will occur.
- An error does not occur even if the value of *Code* is not set as a event code on the Sysmac Studio. If the event code is not registered, the event group and detailed information are not recorded in the user-defined event log. The value of *Code* is recorded for the event name.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE.
  - The value of *Code* is outside of the valid range.
  - An attempt was made to generate more than the maximum number of user-defined errors.

Sample Programming

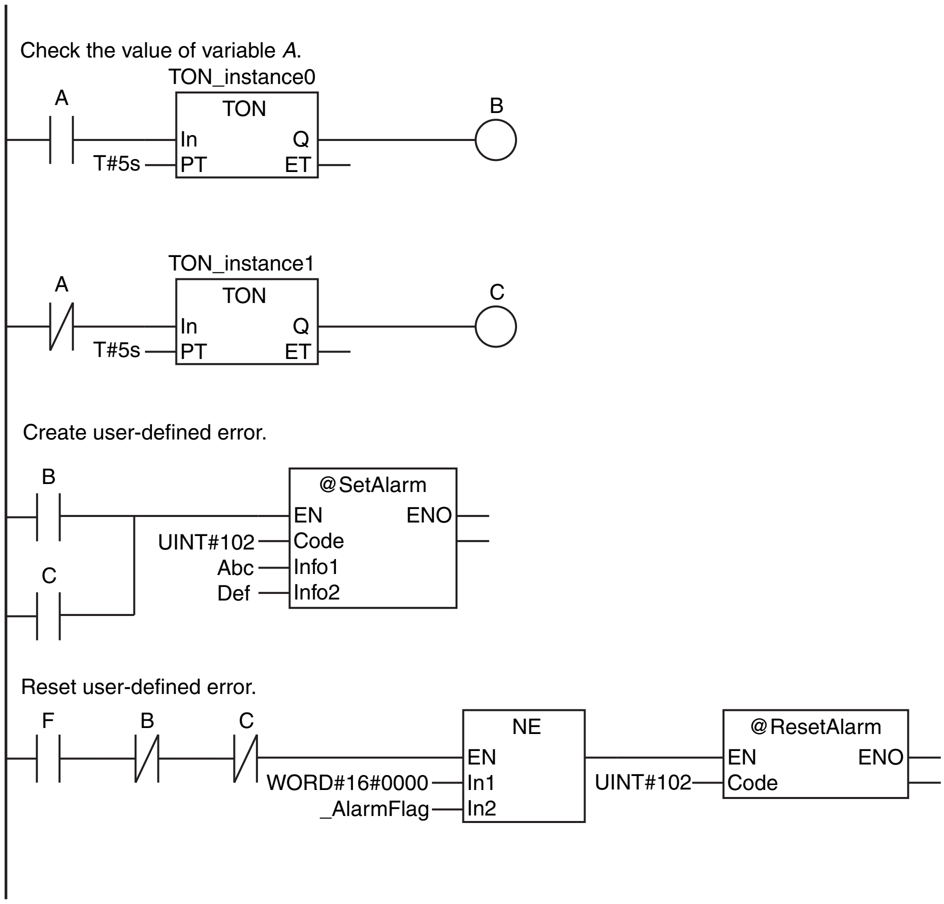
In this sample, the value of variable *A* changes between TRUE and FALSE every five seconds. The value of *A* is monitored. If it does not change for more than five seconds, a user-defined error with event code 102 is generated. UINT#123 and UINT#456 are given as the attached information.

When variable *F* changes to TRUE, the user-defined error is cleared.

LD

Internal Variables	Variable	Data type	Initial value
	A	BOOL	FALSE
	B	BOOL	FALSE
	C	BOOL	FALSE
	F	BOOL	FALSE
	Abc	UINT	123
	Def	UINT	456
	TON_instance0	TON	
	TON_instance1	TON	

External Variables	Variable	Data type	Constant	Comment
	_AlarmFlag	WORD	✔	Error Status of User-defined Errors


Check the value of variable A.
TON_instance0
A
TON
In
Q
T#5s
PT
ET
B
TON_instance1
A
TON
In
Q
T#5s
PT
ET
C
Create user-defined error.
B
@SetAlarm
EN
ENO
UINT#102
Code
Abc
Info1
Def
Info2
C
Reset user-defined error.
F
B
C
NE
EN
WORD#16#0000
In1
_AlarmFlag
In2
UINT#102
@ResetAlarm
EN
ENO
Code

ST

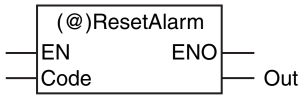
Internal Variables	Variable	Data type	Initial value
	A	BOOL	FALSE
	B	BOOL	FALSE
	C	BOOL	FALSE
	F	BOOL	FALSE
	Abc	UINT	123
	Def	UINT	456
	TON_instance0	TON	
	TON_instance1	TON	

External Variables	Variable	Data type	Constant	Comment
	_AlarmFlag	WORD	✔	Error Status of User-defined Errors

```
// Check the value of variable A.
IF (A=TRUE) THEN
    TON_instance0(In:=TRUE, PT:=T#5s, Q=>B);
ELSE
    TON_instance0(In:=FALSE, Q=>B);
END_IF;

IF (A=FALSE) THEN
    TON_instance1(In:=TRUE, PT:=T#5s, Q=>C);
ELSE
    TON_instance1(In:=FALSE, Q=>C);
END_IF;

// Create user-defined error.
IF (B=TRUE) OR (C=TRUE) THEN
    SetAlarm(
        Code  :=UINT#102
        Info1 :=Abc,
        info2 :=Def);
END_IF;

// Reset user-defined error.
IF (F=TRUE)  & (B=FALSE) & (C=FALSE) & (_AlarmFlag<>WORD#16#0000) THEN
    ResetAlarm(Code:=UINT#102);
END_IF;
```

ResetAlarm

The ResetAlarm instruction resets a user-defined error.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ResetAlarm	Reset User-defined Error	FUN		ResetAlarm(Code);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Code	Event code	Input	Event code of user-defined error to reset 16#0:Reset all application errors.	Depends on data type.	---	0
Out	Return value	Output	Always TRUE	TRUE only	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Code							OK													
Out	OK																			

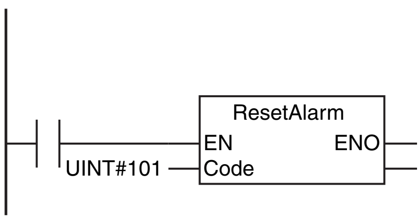
Function

The ResetAlarm instruction resets the user-defined error specified by event code *Code*. An event is then recorded in the user-defined event log area to show that a specific user-defined error was reset. The event code for this event is 65533 and the level is user information.

If the value of *Code* is 0, all current user-defined errors are reset. An event is then recorded in the user-defined event log area to show that all user-defined errors were reset. The event code for this event is 65534 and the level is user information.

The following figure shows a programming example. A user-defined error for event code 101 is reset.

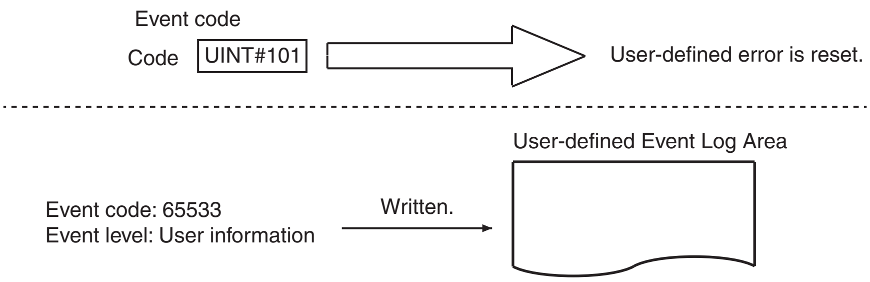
LD



ST

```
ResetAlarm(UINT#101);
```

The ResetAlarm instruction resets the user-defined error specified by event code *Code*.
Also an event is recorded in the user-defined event log area to show that a specific user-defined error was reset.



Related System-defined Variables

Name	Meaning	Data type	Description
_AlarmFlag	Error Status of User-defined Errors	WORD	These flags indicate when user-defined errors are detected. Bit 0 to bit 7 indicate the status of user-defined error levels 1 to 8.*1

*1 Refer to the *NJ/NX-series CPU Unit Software User's Manual* (Cat. No. W501) or *NY-series Industrial Panel PC / Industrial Box PC Software User's Manual* (Cat. No. W558) for details.

Precautions for Correct Use

- An error does not occur if the user-defined error specified by *Code* has not occurred.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following case. *ENO* will be FALSE.
  - The value of *Code* is outside of the valid range.

Sample Programming

Refer to the sample programming that is provided for the SetAlarm instruction (page 2-814).

GetAlarm

The GetAlarm instruction gets the highest event level (of user-defined error levels 1 to 8) and the highest level event code of the current user-defined errors.

Instruction	Name	FB/FUN	Graphic expression	ST expression
GetAlarm	Get User-defined Error Status	FUN	(@)GetAlarm EN — Out Level Code	Out:=GetAlarm(Level, Code);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Out	Error flag	Output	TRUE: User-defined error exists. FALSE: No user-defined error	Depends on data type.	---	---
Level	Highest event level		Highest event level of all current user-defined errors 0: No user-defined error 1 to 8: Event level	0 to 8		
Code	Highest level event code		Highest level event code of all current user-defined errors 0: No user-defined error 1 to 40000: Event level	0 to 40000		

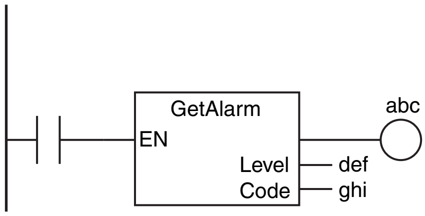
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out	OK																			
Level							OK													
Code							OK													

Function

The GetAlarm instruction gets the highest event level and the highest level event code of the current user-defined errors and outputs them to *Level* and *Code*. If there are currently no user-defined errors, the value of error flag *Out* is FALSE. If there is more than one use-defined error at the highest event level, the value of *Code* is the event code for the user-defined error that occurred first.

The following figure shows a programming example.

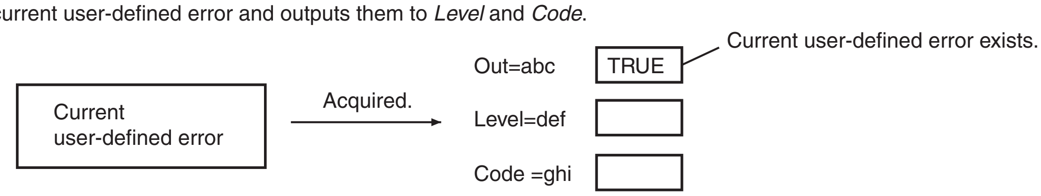
LD



ST

```
abc:=GetAlarm(def, ghi);
```

The GetAlarm instruction gets the highest event level and the highest level event code of the current user-defined error and outputs them to *Level* and *Code*.



Related System-defined Variables

Name	Meaning	Data type	Description
_AlarmFlag	Error Status of User-defined Errors	WORD	These flags indicate when user-defined errors are detected. Bit 0 to bit 7 indicate the status of user-defined error levels 1 to 8.*1

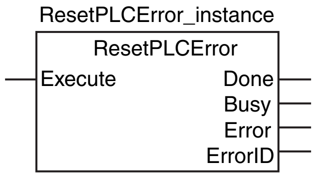
*1 Refer to the *NJ/NX-series CPU Unit Software User's Manual* (Cat. No. W501) or *NY-series Industrial Panel PC / Industrial Box PC Software User's Manual* (Cat. No. W558) for details.

Precautions for Correct Use

If this instruction is used in a ladder diagram, the value of *Out* changes to FALSE if an error occurs in the previous instruction on the rung.

ResetPLCError

The ResetPLCError instruction resets errors in the PLC Function Module.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ResetPLCError	Reset PLC Controller Error	FB		ResetPLCError(Execute, Done, Busy, Error, ErrorID);

Variables

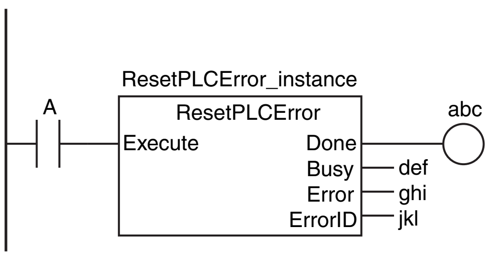
Only common variables are used.

Function

The ResetPLCError instruction resets errors in the PLC Function Module.

The following figure shows a programming example.

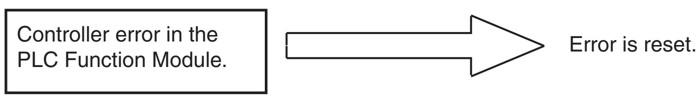
LD



ST

ResetPLCError_instance(A, abc, def, ghi, jkl);

The ResetPLCError instruction resets errors in the PLC Function Module.



Related System-defined Variables

Name	Meaning	Data type	Description
_PLC_ErrSta	Error Status of PLC Function Module	WORD	Contains the error status of the PLC Function Module.*1

*1 Refer to the *NJ/NX-series CPU Unit Software User's Manual* (Cat. No. W501) or *NY-series Industrial Panel PC / Industrial Box PC Software User's Manual* (Cat. No. W558) for details.

Precautions for Correct Use

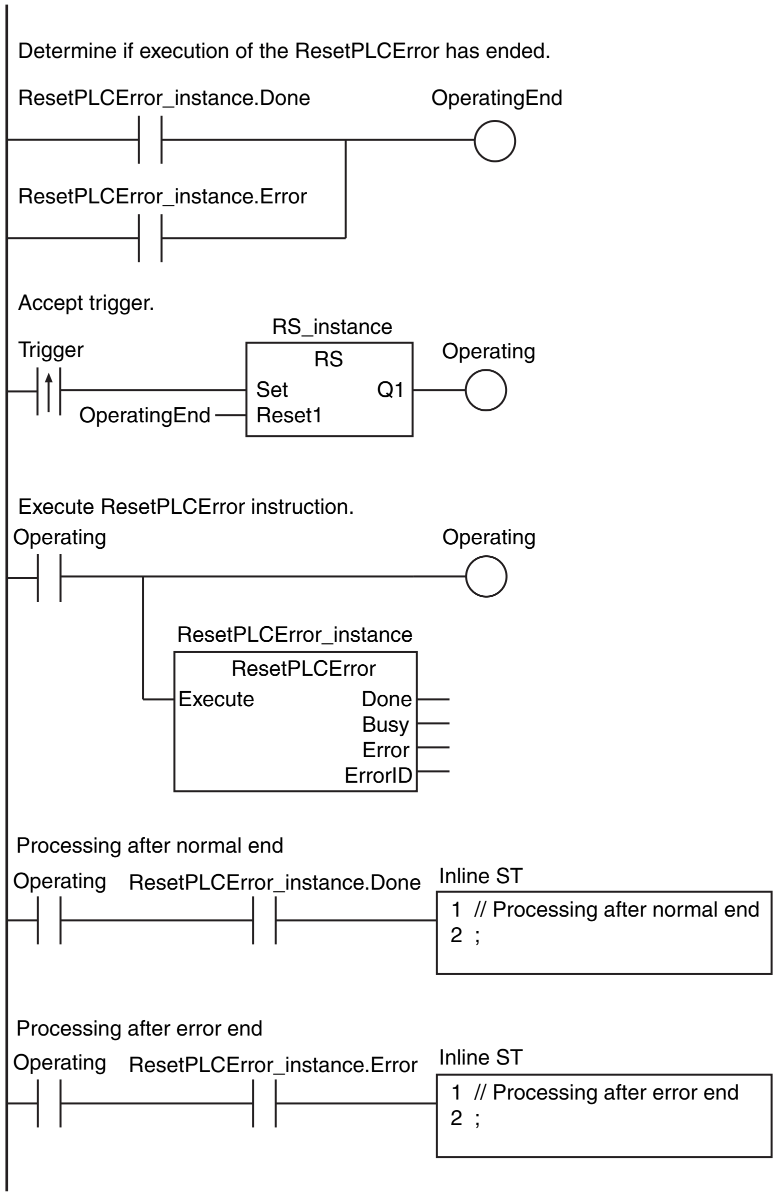
The error may not be reset immediately after you execute this instruction. Use the GetPLCError instruction to confirm that the errors were reset.

Sample Programming

The ResetPLCError instruction is executed when the value of *Trigger* changes to TRUE. Normal end processing is performed if execution of the ResetPLCError instruction ends normally (i.e., if the value of *Done* is TRUE). Error end processing is performed if execution ends in an error (i.e., if the value of *Error* is TRUE).

LD

Variable	Data type	Initial value	Comment
OperatingEnd	BOOL	FALSE	Processing completed
Trigger	BOOL	FALSE	Execution condition
Operating	BOOL	FALSE	Processing
RS_instance	RS		
ResetPLCError_instance	ResetPLCError		



ST

Variable	Data type	Initial value	Comment
Trigger	BOOL	FALSE	Execution condition
LastTrigger	BOOL	FALSE	Value of *Trigger* from previous task period
OperatingStart	BOOL	FALSE	Processing started
Operating	BOOL	FALSE	Processing
ResetPLCError_instance	ResetPLCError		

```
// Detect when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) ) THEN
    OperatingStart:=TRUE;
    Operating     :=TRUE;
END_IF;
LastTrigger:=Trigger;

// Initialize ResetPLCError_instance.
IF (OperatingStart=TRUE) THEN
    ResetPLCError_instance(Execute:=FALSE);
    OperatingStart:=FALSE;
END_IF;

// Execute ResetPLCError instruction.
IF (Operating=TRUE) THEN
    ResetPLCError_instance(Execute:=TRUE);

    IF (ResetPLCError_instance.Done=TRUE) THEN
        // Processing after normal end
        Operating:=FALSE;
    END_IF;

    IF (ResetPLCError_instance.Error=TRUE) THEN
        // Processing after error end
        Operating:=FALSE;
    END_IF;
END_IF;
```

GetPLCError

The GetPLCError instruction gets the highest level status (partial fault or minor fault) and highest level event code of the current Controller errors in the PLC Function Module.

Instruction	Name	FB/FUN	Graphic expression	ST expression
GetPLCError	Get PLC Controller Error Status	FUN	(@)GetPLCError EN — Out Level Code	Out:=GetPLCError(Level, Code);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Out	Error flag	Output	TRUE: Controller error exists. FALSE: No Controller error	Depends on data type.		
Level	Highest level status		Highest level status of all current Controller errors in the PLC Function Module 0: No Controller error 2: Partial fault level 3: Minor fault level	0, 2, or 3		
Code	Highest level event code		Highest level event code of all current Controller errors in the PLC Function Module 16#0000_0000: No Controller error 16#0007_0000 to 16#FFFF_FFFF: Event code	16#00000000 16#00070000 to 16#FFFFFFFF		

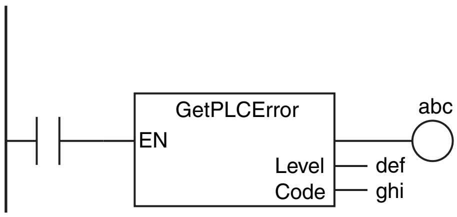
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out	OK																			
Level							OK													
Code				OK																

Function

The GetPLCError instruction gets the highest level status and the highest level event code of the current Controller errors in the PLC Function Module and outputs them to *Level* and *Code*.
If there are currently no Controller errors, the value of error flag *Out* is FALSE.
If there is more than one Controller error at the highest event level, the value of *Code* is the event code for the Controller error that occurred first.

The following figure shows a programming example.

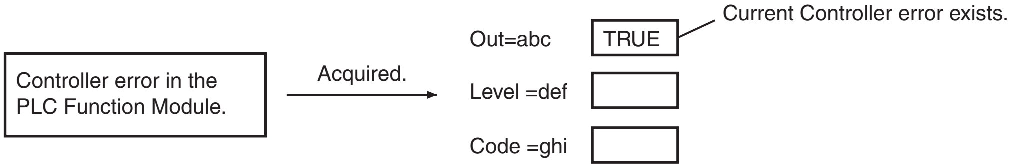
LD



ST

```
abc:=GetPLCError(def, ghi);
```

The GetPLCError instruction gets the highest level status and the highest level event code of the current Controller errors in the PLC Function Module and outputs them to *Level* and *Code*.



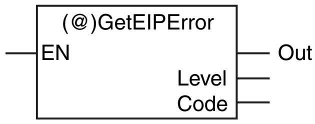
Related System-defined Variables

Name	Meaning	Data type	Description
_PLC_ErrSta	Error Status of PLC Function Module	WORD	Contains the error status of the PLC Function Module.*1

*1 Refer to the *NJ/NX-series CPU Unit Software User's Manual* (Cat. No. W501) or *NY-series Industrial Panel PC / Industrial Box PC Software User's Manual* (Cat. No. W558) for details.

GetEIPError

The GetEIPError instruction gets the highest level status (partial fault or minor fault) and highest level event code of the current Controller errors in the EtherNet/IP Function Module.

Instruction	Name	FB/FUN	Graphic expression	ST expression
GetEIPError	Get EtherNet/IP Error Status	FUN		Out:=GetEIPError(Level, Code);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Out	Error flag	Output	TRUE: Controller error exists. FALSE: No Controller error	Depends on data type.	---	---
Level	Highest event level		Highest level status of all current Controller errors in the EtherNet/IP Function Module 0: No Controller error 2: Partial fault level 3: Minor fault level	0, 2, or 3		
Code	Highest level event code		Highest level event code of all current Controller errors in the EtherNet/IP Function Module 16#0000_0000: No Controller error 16#0007_0000 to 16#FFFF_FFFF: Event code	16#00000000 16#00070000 to 16#FFFFFFFF		

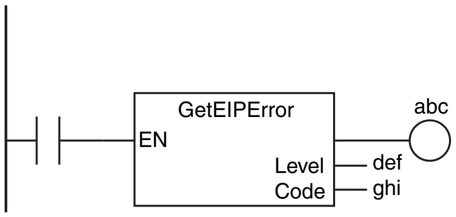
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out	OK																			
Level							OK													
Code				OK																

Function

The GetEIPError instruction gets the highest level status and the highest level event code of the current Controller errors in the EtherNet/IP Function Module and outputs them to *Level* and *Code*. If there are currently no Controller errors, the value of error flag *Out* is FALSE. If there is more than one Controller error at the highest event level, the value of *Code* is the event code for the Controller error that occurred first.

The following figure shows a programming example.

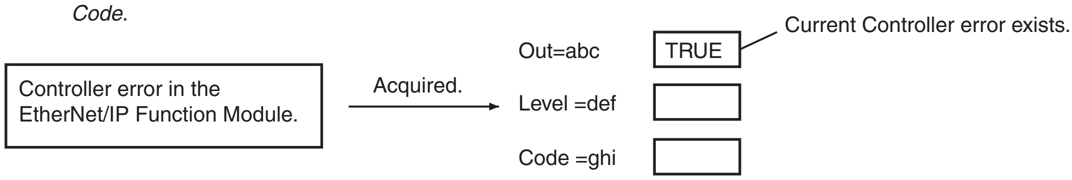
LD



ST

```
abc:=GetEIPError(def, ghi);
```

The GetEIPError instruction gets the highest level status and the highest level event code of the current Controller errors in the EtherNet/IP Function Module and outputs them to *Level* and *Code*.



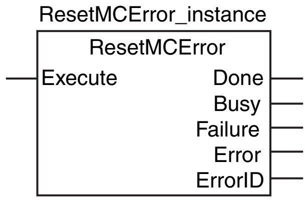
Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_ErrSta	Error Status of EtherNet/IP Function Module	WORD	Contains the error status of the EtherNet/IP Function Module.*1

*1 Refer to the *NJ/NX-series CPU Unit Software User's Manual* (Cat. No. W501) or *NY-series Industrial Panel PC / Industrial Box PC Software User's Manual* (Cat. No. W558) for details.

ResetMCError

The ResetMCError instruction resets Controller errors in the Motion Control Function Module.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ResetMCError	Reset Motion Control Error	FB		ResetMCError_instance(Execute, Done, Busy, Failure Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Failure	Failure end	Output	TRUE: The errors were not reset. FALSE: The errors were reset normally.	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Failure	OK																			

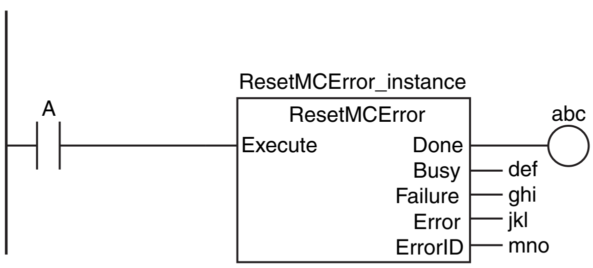
Function

The ResetMCError instruction resets a Controller error in the Motion Control Function Module.
If the errors are not reset, the value of *Failure* changes to TRUE.

No matter what task the program that executes the ResetMCError is placed in, this instruction resets errors for all axes and all axes groups.

The following figure shows a programming example.

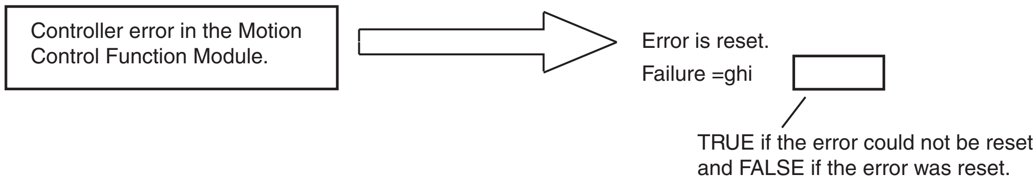
LD



ST

```
ResetMCError_instance(A, abc, def,
                      ghi, jkl, mno);
```

The ResetMCError instruction resets Controller errors in the Motion Control Function Module.
If the errors are not reset, the value of *Failure* changes to TRUE.



Related System-defined Variables

Name	Meaning	Data type	Description
_MC_ErrSta	Motion Control Error Status	WORD	Contains the error status of the Motion Control Function Module.*1

*1 Refer to the *NJ/NX-series CPU Unit Software User's Manual* (Cat. No. W501) or *NY-series Industrial Panel PC / Industrial Box PC Software User's Manual* (Cat. No. W558) for details.

Precautions for Correct Use

- The error may not be reset immediately after you execute this instruction. Use the GetMCError instruction to confirm that the errors were reset.
- If you attempt to execute this instruction during an MC Test Run, the value of *Busy* remains TRUE and the instruction is not executed.
- If you execute this instruction for an OMRON G5-series Servo Drive, perform exclusive control of the instructions so that the ResetECError instruction is not executed at the same time.
  If the ResetMCError and ResetECError instructions are executed at the same time, the G5-series Servo Drive will no longer accept SDO communications.

Version Information

- With a CPU Unit with unit version 1.02 to 1.09, you can create only 100 instances of this instruction.
- If you transfer a user program that has more than 100 instances of this instruction to a Controller with a CPU Unit with unit version 1.02 to 1.09, a Controller error will occur. The Controller error depends on the transfer method that is used for the user program as given below.

User program transfer method	Event code for Controller error	Level of Controller error
Project transferred with synchronization function	10250000 hex	Major fault level
	571D0000 hex	Observation
User program transferred with online editing	571D0000 hex	Observation

- If you transfer a user program that has more than 100 instances of this instruction to a Controller with a CPU Unit with unit version 1.01 or earlier, the above Controller error will not occur. However, if you create too many instances of this instruction, the user program will become too large and a major fault level Controller error will occur.

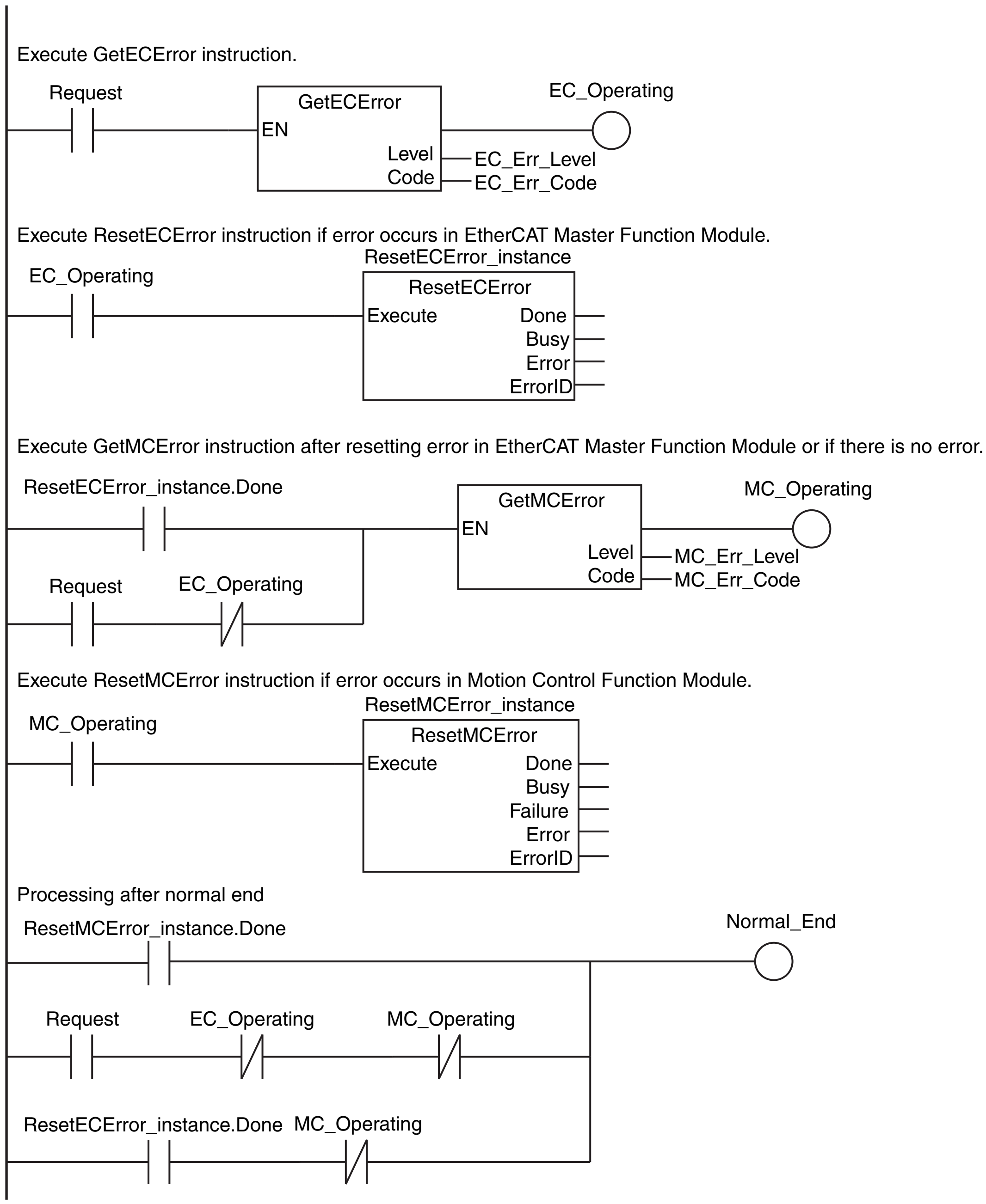
Sample Programming

This sample detects Controller errors in the EtherCAT Master Function Module and Motion Control Function Module. If errors are detected, they are reset. The processing procedure is as follows:

1. The GetECError instruction is executed to detect any Controller errors in the EtherCAT Master Function Module.
2. If errors are detected, they are reset with the ResetECError instruction.
3. The GetMCError instruction is executed to detect any Controller errors in the Motion Control Function Module.
4. If errors are detected, they are reset with the ResetMCError instruction.

LD

Variable	Data type	Initial value	Comment
Request	BOOL	FALSE	Error detection reset request
EC_Err_Level	UINT	0	Highest event level in EtherCAT Master Function Module
EC_Err_Code	DWORD	DWORD#16#0	Highest level event code in EtherCAT Master Function Module
EC_Operating	BOOL	FALSE	Resetting error in EtherCAT Master Function Module
MC_Err_Level	UINT	0	Highest event level in Motion Control Function Module
MC_Err_Code	DWORD	DWORD#16#0	Highest level event code in Motion Control Function Module
MC_Operating	BOOL	FALSE	Resetting error in Motion Control Function Module
Normal_End	BOOL	FALSE	Normal end
ResetECError_instance	ResetECError		
ResetMCError_instance	ResetMCError		



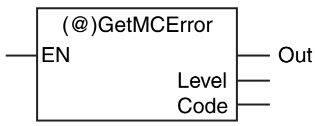
ST

Variable	Data type	Initial value	Comment
Request	BOOL	FALSE	Error detection reset request
EC_Error	BOOL	FALSE	Error in EtherCAT Master Function Module
EC_Err_Level	UINT	0	Highest event level in EtherCAT Master Function Module
EC_Err_Code	DWORD	DWORD#16#0	Highest level event code in EtherCAT Master Function Module
EC_Stage	INT	0	Error reset in EtherCAT Master Function Module
MC_Error	BOOL	FALSE	Error in Motion Control Function Module
MC_Err_Level	UINT	0	Highest event level in Motion Control Function Module
MC_Err_Code	DWORD	DWORD#16#0	Highest level event code in Motion Control Function Module
MC_Stage	INT	0	Error reset in Motion Control Function Module
ResetECError_instance	ResetECError		
ResetMCError_instance	ResetMCError		

```
IF (Request=TRUE) THEN                                  // Determine error resetting requests.
     EC_Error:=GetECError(EC_Err_Level, EC_Err_Code);   // Detect Controller errors in EtherCAT Master Function
                                                        // Module.
     MC_Error:=GetMCError(MC_Err_Level, MC_Err_Code);   // Detect Controller errors in Motion Control Function
                                                        // Module.
     IF (EC_Error=TRUE) THEN  // Controller error in EtherCAT Master Function Module.
          CASE EC_Stage OF
          0 :          // Initialize
               ResetECError_instance(Execute:=FALSE);
               EC_Stage:=INT#1;
          1 :          // Resetting Controller error in EtherCAT Master Function Module.
               ResetECError_instance(Execute:=TRUE);
               IF (ResetECError_instance.Done=TRUE) THEN
                    EC_Stage:=INT#99;    // Normal end
               END_IF;
               IF (ResetECError_instance.Error=TRUE) THEN
                    EC_Stage:=INT#98;    // Error end
               END_IF;
          99 :         // Processing after normal end
               EC_Stage:=INT#0;
          98 :         // Processing after error end.
               EC_Stage:=INT#0;
          END_CASE;
     END_IF;
     IF (MC_Error=TRUE) THEN   // Controller error in Motion Control Function Module.
          CASE MC_Stage OF
          0 :          // Initialize
               ResetMCError_instance(Execute:=FALSE);
               MC_Stage:=INT#1;
          1 :          // Resetting Controller error in Motion Control Function Module.
               IF (EC_Error=FALSE) THEN    // Recover operation for all slaves.
                    ResetMCError_instance(Execute:=TRUE);
                    IF (ResetMCError_instance.Done=TRUE) THEN
                         MC_Stage:=INT#99;   // Normal end
                    END_IF;
                    IF ( (ResetMCError_instance.Error=TRUE) OR (ResetMCError_instance.Failure=TRUE) ) THEN
                         MC_Stage:=INT#98;   // Error end
                    END_IF;
               END_IF;
          99 :       // Processing after normal end
               MC_Stage:=INT#0;
          98 :       // Processing after error end.
               MC_Stage:=INT#0;
          END_CASE;
     END_IF;
END_IF;
```

GetMCError

The GetMCError instruction gets the highest level status (partial fault or minor fault) and highest level event code of the current Controller errors in the Motion Control Function Module.

Instruction	Name	FB/FUN	Graphic expression	ST expression
GetMCError	Get Motion Control Error Status	FUN		Out:=GetMCError(Level, Code);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Out	Error flag	Output	TRUE: Controller error exists. FALSE: No Controller error	Depends on data type.	---	---
Level	Highest level status		Highest level status of all current Controller errors in the Motion Control Function Module 0: No Controller error 2: Partial fault level 3: Minor fault level	0, 2, or 3		
Code	Highest level event code		Highest level event code of all current Controller errors in the Motion Control Function Module 16#0000_0000: No Controller error 16#0007_0000 to 16#FFFF_FFFF: Event code	16#00000000 16#00070000 to 16#FFFFFFFF		

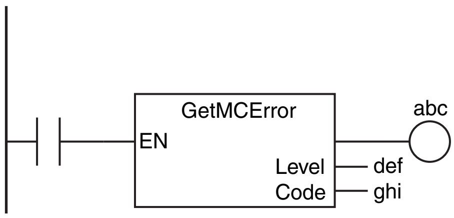
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out	OK																			
Level							OK													
Code				OK																

Function

The GetMCError instruction gets the highest level status and the highest level event code of the current Controller errors in the Motion Control Function Module and outputs them to *Level* and *Code*. If there are currently no Controller errors, the value of error flag *Out* is FALSE. If there is more than one Controller error at the highest event level, the value of *Code* is the event code for the Controller error that occurred first.

The following figure shows a programming example.

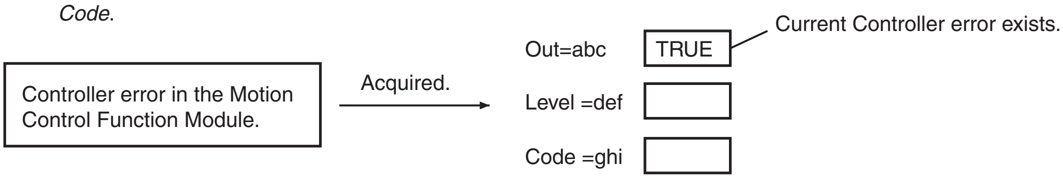
LD



ST

```
abc:=GetMCError(def, ghi);
```

The GetMCError instruction gets the highest level status and the highest level event code of the current Controller errors in the Motion Control Function Module and outputs them to *Level* and *Code*.



Related System-defined Variables

Name	Meaning	Data type	Description
_MC_ErrSta	Error Status of Motion Control Function Module	WORD	Contains the error status of the Motion Control Function Module.*1

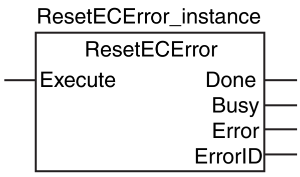
*1 Refer to the *NJ/NX-series CPU Unit Software User's Manual* (Cat. No. W501) or *NY-series Industrial Panel PC / Industrial Box PC Software User's Manual* (Cat. No. W558) for details.

Sample Programming

Refer to the sample programming that is provided for the ResetMCError instruction (page 2-830).

ResetECError

The ResetECError instruction resets a Controller error in the EtherCAT Master Function Module.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ResetECError	Reset EtherCAT Error	FB		ResetECError_instance(Execute, Done, Busy, Error, ErrorID);

Variables

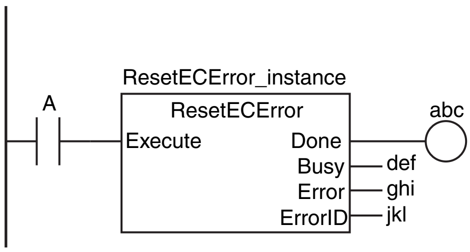
Only common variables are used.

Function

The ResetECError instruction resets Controller errors in the EtherCAT Master Function Module.

The following figure shows a programming example.

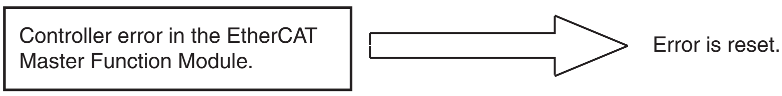
LD



ST

```
ResetECError_instance(A, abc, def, ghi, jkl);
```

The ResetECError instruction resets a Controller error in the EtherCAT Master Function Module.



Related System-defined Variables

Name	Meaning	Data type	Description
_EC_ErrSta	Built-in EtherCAT Error	WORD	Contains a summary of the errors in the EtherCAT Master Function Module.*1

*1 Refer to the *NJ/NX-series CPU Unit Software User's Manual* (Cat. No. W501) or *NY-series Industrial Panel PC / Industrial Box PC Built-in EtherCAT Port User's Manual* (Cat. No. W562) for details.

Precautions for Correct Use

- The error may not be reset immediately after you execute this instruction. Use the GetECError instruction to confirm that the errors were reset.
- If you execute this instruction for an OMRON G5-series Servo Drive, perform exclusive control of the instructions so that the ResetMCError, MC_Reset, or MC_GroupReset instruction is not executed at the same time. If any of these three instructions and the ResetECError instruction are executed at the same time, the G5-series Servo Drive will no longer accept SDO communications.
- You cannot execute this instruction during execution of the following instructions: EC_Disconnect-Slave, EC_ConnectSlave, EC_ChangeEnableSetting, ResetECError, RestartNXUnit, and NX_ChangeWriteMode.
- An error occurs in the following case. *Error* will change to TRUE.
    - This instruction is executed again while processing to clear a Controller error from the EtherCAT Master Function Module is in progress.
    - The EC_DisconnectSlave, EC_ConnectSlave, EC_ChangeEnableSetting, ResetECError, RestartNXUnit, or NX_ChangeWriteMode instruction is already in execution.

Sample Programming

Refer to the sample programming that is provided for the ResetMCError instruction (page 2-830).

GetECError

The GetECError instruction detects errors in the EtherCAT Master Function Module.

Instruction	Name	FB/FUN	Graphic expression	ST expression
GetECError	Get EtherCAT Error Status	FUN	(@)GetECError EN — Out Level Code	Out:=GetECError(Level, Code);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Out	Error flag	Output	TRUE: Error exists.*1 FALSE: No error	Depends on data type.	---	---
Level	Highest level status		Status of the current error with the highest level*1 0: No error 2: Partial fault level 3: Minor fault level	0, 2, or 3		
Code	Highest level event code		Event code of the current error with the highest level*1	16#00000000 16#00070000 to 16#FFFFFFFF		

*1 The errors that are detected depend on the unit version of the CPU Unit and the version of the Sysmac Studio. Refer to *Function* for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out	OK																			
Level							OK													
Code				OK																

Function

The GetECError instruction detects errors in the EtherCAT Master Function Module.

The value of *Out* is TRUE if there is an error and FALSE if there is no error.

Level gives the status of the current error with the highest level.

Code gives the event code of the current error with the highest level.

Detected Errors and Output Variable Values

The errors that are detected by this instruction depend on the unit version of the CPU Unit. The following table lists the errors that are detected for each unit version.

Unit version of CPU Unit	Detected errors
1.02 or later	Communications port errors, master errors, and slave errors
1.01 or earlier	Communications port errors and master errors

The following table shows the relationship between the unit version of the CPU Unit, the status of the EtherCAT Master Function Module, and the values of the output variables.

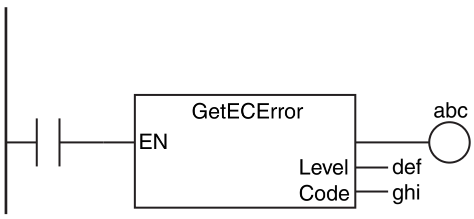
<table>
<tr><th>Unit version of CPU Unit</th><th>Status of EtherCAT Master Function Module</th><th>Value of Out</th><th>Value of Level</th><th>Value of Code</th></tr>
<tr><td rowspan="3">1.04 or later</td><td>No error</td><td>FALSE</td><td>0</td><td>16#0000_0000</td></tr>
<tr><td>Communications port error or master error</td><td rowspan="2">TRUE</td><td rowspan="2">Status of the current error with the highest level
2: Partial fault level
3: Minor fault level</td><td>Event code of the current error with the highest level.*1
16#0007_0000 to 16#FFFF_FFFF</td></tr>
<tr><td>Slave error</td><td>16#0000_0000</td></tr>
<tr><td rowspan="3">1.02 or 1.03</td><td>No error</td><td>FALSE</td><td>0</td><td>16#0000_0000</td></tr>
<tr><td>Communications port error or master error</td><td>TRUE</td><td rowspan="2">Status of the current error with the highest level
2: Partial fault level
3: Minor fault level</td><td>Event code of the current error with the highest level.*1
16#0007_0000 to 16#FFFF_FFFF</td></tr>
<tr><td>Slave error</td><td>FALSE</td><td>16#0000_0000</td></tr>
<tr><td rowspan="3">1.00 or 1.01</td><td>No error</td><td>FALSE</td><td>0</td><td>16#0000_0000</td></tr>
<tr><td>Communications port error or master error</td><td>TRUE</td><td>Status of the current error with the highest level
2: Partial fault level
3: Minor fault level</td><td>Event code of the current error with the highest level.*1
16#0007_0000 to 16#FFFF_FFFF</td></tr>
<tr><td>Slave error</td><td>FALSE</td><td>0</td><td>16#0000_0000</td></tr>
</table>

*1 If there is more than one error at the highest event level, the value of *Code* is the event code for the error that occurred first.

Notation Example

The following figure shows a programming example.

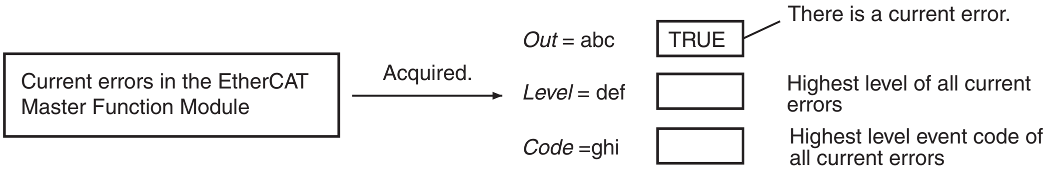
LD



ST

```
abc:=GetECError(def, ghi);
```

The GetECError instruction detects current communications port errors, master errors, and slave errors in the EtherCAT Master Function Module.



Related System-defined Variables

Name	Meaning	Data type	Description
_EC_ErrSta	Built-in EtherCAT Error	WORD	Contains a summary of the errors in the EtherCAT Master Function Module.*2
_EC_PortErr*1	Communications Port Error	WORD	Contains a summary of the EtherCAT master communications port errors.*2
_EC_MstrErr*1	Master Error	WORD	Contains a summary of the EtherCAT master errors and the slave errors detected by the EtherCAT master.*2
_EC_SlavErr	Slave Error	WORD	Contains a summary of the overall EtherCAT slave error status.*2

*1 The GetECError instruction gets the errors that are shown by _EC_PortErr (Communications Port Error) and _EC_MstrErr (Master Error).

*2 Refer to the *NJ/NX-series CPU Unit Built-in EtherCAT Port User's Manual* (Cat. No. W505) or *NY-series Industrial Panel PC / Industrial Box PC Built-in EtherCAT Port User's Manual* (Cat. No. W562) for details.

Precautions for Correct Use

Version Information

A CPU Unit with unit version 1.02 or later is required to detect slave errors with this instruction.

Sample Programming

Refer to the sample programming that is provided for the ResetMCError instruction (page 2-830).

SetInfo

The SetInfo instruction creates user-defined information.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SetInfo	Create User-defined Information	FUN	(@)SetInfo EN ENO Code Out Info1 Info2	SetInfo(Code, Info1, Info2);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Code	Event code	Input	Event code of user-defined information to generate	40001 to 60000	---	40001
Info1	Attached information 1		Values recorded in event log when the user-defined information is generated	Depends on data type.		*
Info2	Attached information 2					
Out	Return value	Output	Always TRUE	TRUE only	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

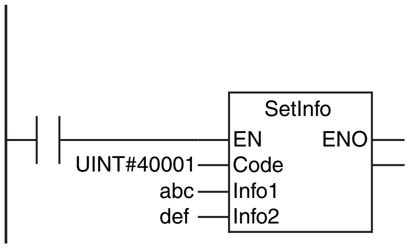
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Code							OK													
Info1	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	
Info2	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	
Out	OK																			

Function

The SetInfo instruction generates the user-defined information specified by event code *Code*. The time of occurrence, event code *Code*, event level, attached information *Info1*, and attached information *Info2* are stored in the user event log area that corresponds to the level of the event code.

The following figure shows a programming example. User-defined information for event code 40001 is generated. The values of variables *abc* and *def* are stored as attached information.

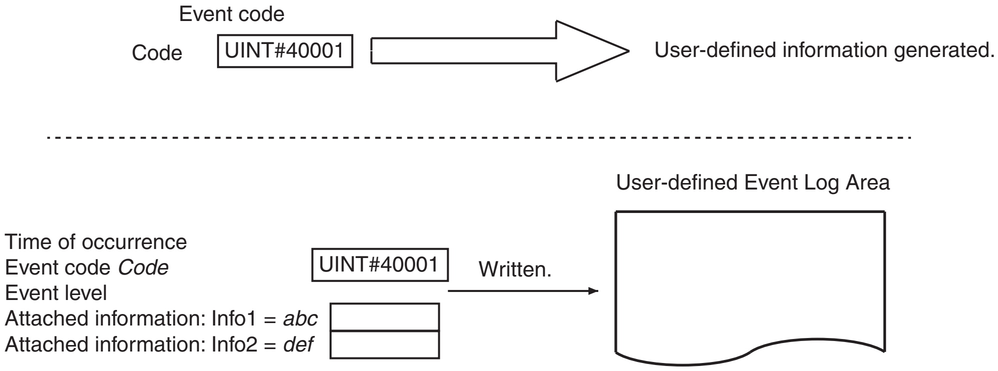
LD



ST

```
SetInfo(UINT#40001, abc, def);
```

The SetInfo instruction generates the user-defined information specified by event code *Code*. Also, the time of occurrence, event code *Code*, event level, attached information *Info1*, and attached information *Info2* are stored in the user event log area that corresponds to the level of the event code.

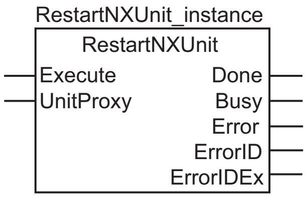


Precautions for Correct Use

- Always use variables for the input parameters that are passed to *Info1* and *Info2*. If the attached information is not used, specify a dummy variable. A building error will occur if a constant is specified.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following case. *ENO* will be FALSE.
  - The value of *Code* is outside of the valid range.

RestartNXUnit

The RestartNXUnit instruction restarts an EtherCAT Coupler Unit or NX Unit.

Instruction	Name	FB/FUN	Graphic expression	ST expression
RestartNXUnit	Restart NX Units	FB		RestartNXUnit_instance(Execute, UnitProxy, Done, Busy, Error, ErrorID, ErrorIDEx);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
UnitProxy	Specified Unit	Input	A Unit to restart: EtherCAT Coupler Unit, NX Bus Function Module or NX Unit	---	---	*

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
UnitProxy	Refer to *Function* for details on the structure _sNXUNIT_ID.																			

Function

The RestartNXUnit instruction restarts an EtherCAT Coupler Unit or an NX Unit on the EtherCAT Coupler Unit, and an NX Unit connected to the NX bus on the NX Bus Function Module or on the CPU Unit.
You can use it to restart a specified Unit independently.
However, you cannot restart an EtherCAT Coupler Unit or NX Bus Function Module independently.
If you specify an EtherCAT Coupler Unit or NX Bus Function Module, all of the NX Units that are connected to it are also restarted.
The Unit to restart is specified with *UnitProxy.*

The data type of *UnitProxy* is structure _sNXUNIT_ID. The meanings of the members are as follows:

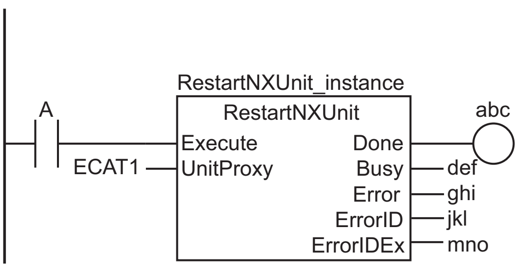
Name	Meaning	Content	Data type
UnitProxy	Specified Unit	Specified Unit	_sNXUNIT_ID
NodeAdr	Node address	Node address of the Communications Coupler Unit	UINT
IPAdr	IP address	IP address of the Communications Coupler Unit	BYTE[5]
UnitNo	Unit number	Unit number of specified Unit	UDINT
Path	Path	Path information to the specified Unit	BYTE[64]
PathLength	Valid path length	Valid path length	USINT

Pass a device variable that is assigned to the specified EtherCAT Coupler Unit or an NX Unit on the EtherCAT Coupler Unit, and an NX Unit connected to the NX bus on the NX Bus Function Module or on the CPU Unit to *UnitProxy*.

Notation Example

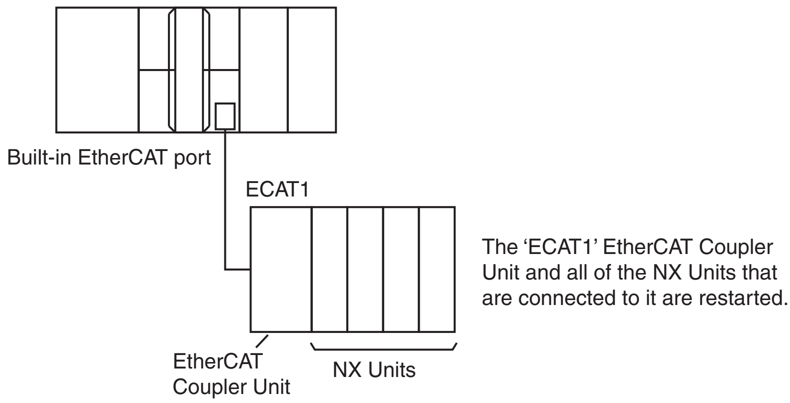
The following example shows a case of restarting all EtherCAT Slave Terminals. A variable that is named ‘ECAT1’ with a data type of _sNXUNIT_ID is assigned to the EtherCAT Coupler Unit.

LD



ST

```
RestartNXUnit_instance(A, ECAT1,
                       abc, def, ghi, jkl, mno);
```



Related System-defined Variables

Name	Meaning	Data type	Description
_EC_MBXSlavTbl[i] "i" is the node address.	Message Communications Enabled Slave Table	BOOL	This variable indicates whether communications are possible for each slave. TRUE: Communications are possible. FALSE: Communications are not possible.
_NXB_UnitMsgActiveTbl[i]	NX Unit Message Enabled Status	BOOL	This table indicates the slaves that can perform message communications. Use this variable to confirm that communications with the relevant slave are possible.

Additional Information

You can use the following procedure to write data with the following attributes to an EtherCAT Coupler Unit, an NX Unit on the EtherCAT Coupler Unit, or an NX Unit connected to the NX bus of the CPU Unit.

- Power OFF Retain attribute
- The values are updated when the Unit is restarted.

1 Use the NX_ChangeWriteMode instruction (page 2-851) to change the Unit to a mode that allows writing data.

2 Use the NX_WriteObj instruction (page 2-954) to write data to the Unit.

3 Use the NX_SaveParam instruction (page 2-856) to save the data that you wrote.

4 Use the RestartNXUnit instruction to restart the Unit.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* (page *2-3*) for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- If you specify a Unit that is assigned to a motion control axis (data type _sAXIS_REF) for *UnitProxy*, a Controller error will occur in the Motion Control Function Module. Use the ResetMCError instruction (page 2-830) to reset the Controller error.
- For *UnitProxy,* specify the device variable that is assigned to the EtherCAT Coupler Unit or an NX Unit on the EtherCAT Coupler Unit, and an NX Unit connected to the NX bus of the NX Bus Function Module or the CPU Unit in the I/O Map of the Sysmac Studio. Refer to the *Sysmac Studio Version 1 Operation Manual* (Cat. No. W504-E1-07 or later) for details on assigning device variables.
- If the RestartNXUnit instruction is executed during execution of another RestartNXUnit instruction or execution of the NX_ChangeWriteMode instruction (page 2-851), the RestartNXUnit instruction that is executed later will be queued. Up to 192 instructions can be queued. A building error will occur if an attempt is made to queue more than 192 instructions. The time that an instruction is queued is not included in the timeout time.
- The value of *Busy* is TRUE while the instruction is queued.
- This instruction is related to NX Message Communications Errors. If too many instructions that are related to NX Message Communications Errors are executed at the same time, an NX Message Communications Error will occur. Refer to *A-3 Instructions Related to NX Message Communications Errors* for a list of the instructions that are related to NX Message Communications Errors.
- You cannot execute this instruction during execution of the following instructions: EC_Disconnect-Slave, EC_ConnectSlave, EC_ChangeEnableSetting, ResetECError, RestartNXUnit, and NX_ChangeWriteMode. An error will occur if you attempt to execute it.

- *Error* is TRUE if an error occurred. The meanings of the values of *ErrorID* and *ErrorIDEx* are given in the following table.

Value of *ErrorID*	Value of *ErrorIDEx*	Meaning
16#0419	16#0000 0000	The data type of *UnitProxy* is not correct.
16#2C00	16#0000 0401	The specified Unit does not support the instruction.
	16#0000 1001 16#0000 1002 16#0017 0000 16#0020 0000 16#0021 0000	An input parameter, output parameter, or in-out parameter is incorrect. Confirm that the intended parameter is used for the input parameter, output parameter, or in-out parameter.
	16#0000 1010	The data size of the specified NX object does not agree with the data size specified in *WriteDat*.
	16#0000 1101	The Unit is not correct. Check the Unit.
	16#0000 110B	The size of the read data is too large. Make sure that the read data specification is correct.
	16#0000 1110	There is no object that corresponds to the value of *Obj.Index.*
	16#0000 1111	There is no object that corresponds to the value of *Obj.Subindex.*
	16#0000 2101	The specified NX object cannot be written.
	16#0000 2110	The value of *WriteDat* exceeds the range of the values of the NX object to write.
	16#0000 2210	The specified Unit is not in a mode that allows writing data.
	16#0000 2213	Instruction execution was not possible because the specified Unit was performing an I/O check. Execute the instruction after the I/O check is completed.
	16#0000 2230	The status of the specified Unit does not agree with the value of the read source or write destination NX object. Take the following actions if the value of *Obj.Index* is between 0x6000 and 0x6FFF or between 0x7000 and 0x7FFF. • Delete the read source or write designation NX object from the I/O allocation settings. • Reset the error for the specified Unit. • Place the specified Unit in a mode that does not allow writing data.
	16#0000 2231	Instruction execution was not possible because the specified Unit was performing initialization. Wait for the Unit to start normal operation and then execute the instruction.
	16#0000 250F	Hardware access failed. Execute the instruction again.
	16#0000 2601 16#0000 2602 16#0010 0000	The specified Unit does not support this instruction. Check the version of the Unit.
	16#0000 2603	Execution of the instruction failed. Execute the instruction again. Make sure that at least one channel is enabled in the selections of the channels to use.
	16#0000 2621	The NX Unit is not in a status in which it can acknowledge the instruction. Wait for a while and then execute the instruction again.
	16#0001 0000	The specified Unit does not exist. Make sure that the Unit configuration is correct.
	16#0011 0000	The specified port number does not exist. Make sure that the Unit configuration is correct.
	16#0012 0000 16#0013 0000 16#0015 0000 16#0016 0000	The value of *UnitProxy* is not correct. Set the variable that indicates the specified EtherCAT Coupler Unit again.
	16#0014 0000	The specified node address is not correct. Make sure that the Unit configuration is correct.

Value of *ErrorID*	Value of *ErrorIDEx*	Meaning
16#2C00	16#0030 0000 16#8001 0000	The specified Unit is busy. Execute the instruction again.
	16#0031 0000	The specified Unit is not supported for connection. Check the version of the Unit.
	16#80000000 16#80050000 16#81010000 16#81020000 16#82020000 16#82030000 16#82060000 to 16#8FFF0000 16#90010000 to 16#FFFE0000	An error occurred in the communications network. Execute the instruction again.
	16#8002 0000 16#8003 0000 16#8103 0000 16#8200 0000	An error occurred in the communications network. Reduce the amount of communications traffic.
	16#8004 0000 16#8100 0000 16#8201 0000 16#8204 0000 16#8205 0000 16#9000 0000	An error occurred in the communications network. Check the Unit and cable connections. Make sure that the power supply to the Unit is ON.
16#2C01	16#0000 0000	An attempt was made to queue more than 192 RestartNXUnit and NX_ChangeWriteMode instructions.
16#2C02	16#0000 0000	A timeout occurred during communications.
16#2C05	---	An error occurred in the EtherCAT network. Check the value of *UnitProxy* and the EtherCAT network configuration.
16#2C06	16#00000000	The specified Unit is already being restarted from the Sysmac Studio. Therefore, this instruction does not need to be executed.
16#2C07	16#0000 0000	A slave that cannot be specified for the instruction was connected at the slave node address of the specified Unit. Check the value of *UnitProxy* and the EtherCAT network configuration.

Version Information

A CPU Unit with unit version 1.05 or later and Sysmac Studio version 1.06 or higher are required to use this instruction. However, some versions/unit versions of the following products do not support restarting specified NX Units independently.

- CPU Units
- Sysmac Studio
- EtherCAT Coupler Units
- NX Units

If the unit version of a product does not support restarting specified NX Units independently, you can specify only the EtherCAT Coupler Unit as the Unit to restart.

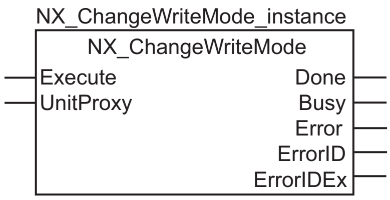
Refer to the *NX-series EtherCAT Coupler Unit User's Manual* (Cat. No. W519-E1-03 or later) for the unit versions of products that support restarting specified NX Units independently.

Sample Programming

Refer to the sample programming for the NX_WriteObj instruction (page 2-954).

NX_ChangeWriteMode

The NX_ChangeWriteMode instruction changes an EtherCAT Coupler Unit or NX Unit to a mode that allows writing data.

Instruction	Name	FB/FUN	Graphic expression	ST expression
NX_Change WriteMode	Change to NX Unit Write Mode	FB		NX_ChangeWriteMode_instance (Execute, UnitProxy, Done, Busy, Error, ErrorID, ErrorIDEx);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
UnitProxy	Specified Unit	Input	Unit for which to change the mode	---	---	*

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
UnitProxy	Refer to *Function* for details on the structure _sNXUNIT_ID.																			

Function

The NX_ChangeWriteMode instruction changes the mode for an EtherCAT Coupler Unit, an NX Unit on the EtherCAT Coupler Unit, or an NX Unit connected to the NX bus of the CPU Unit to a mode that allows writing data. The Unit for which to change the mode is specified with *UnitProxy*. Data can be written when the value of *Done* changes to TRUE.

The data type of *UnitProxy* is structure _sNXUNIT_ID. The meanings of the members are as follows:

Name	Meaning	Content	Data type
UnitProxy	Specified Unit	Unit for which to change the write mode	_sNXUNIT_ID
NodeAdr	Node address	Node address of the Communications Coupler Unit	UINT
IPAdr	IP address	IP address of the Communications Coupler Unit	BYTE[5]
UnitNo	Unit number	Unit number of specified Unit	UDINT
Path	Path	Path information to the specified Unit	BYTE[64]
PathLength	Valid path length	Valid path length	USINT

Pass the device variable that is assigned to the specified Unit to *UnitProxy*.

Related Instructions and Execution Procedure

You can use this instruction to write data with the following attributes to an EtherCAT Coupler Unit, an NX Unit on the EtherCAT Coupler Unit, or an NX Unit connected to the NX bus of the CPU Unit.

- Power OFF Retain attribute
- The values are updated when the Unit is restarted.

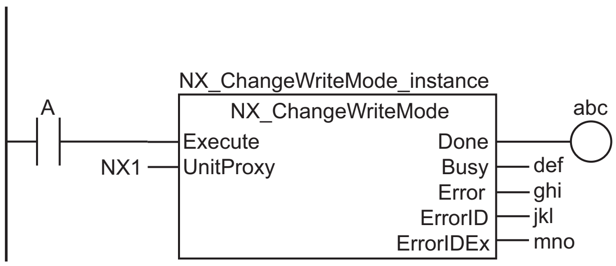
Use the following procedure to execute the related instructions.

1. Use the NX_ChangeWriteMode instruction to change the Units to a mode that allows writing data.
2. Use the NX_WriteObj instruction (page 2-954) to write data to the Unit.
3. Use the NX_SaveParam instruction (page 2-856) to save the data that you wrote.
4. Use the RestartNXUnit instruction (page 2-844) to restart the Unit.

Notation Example

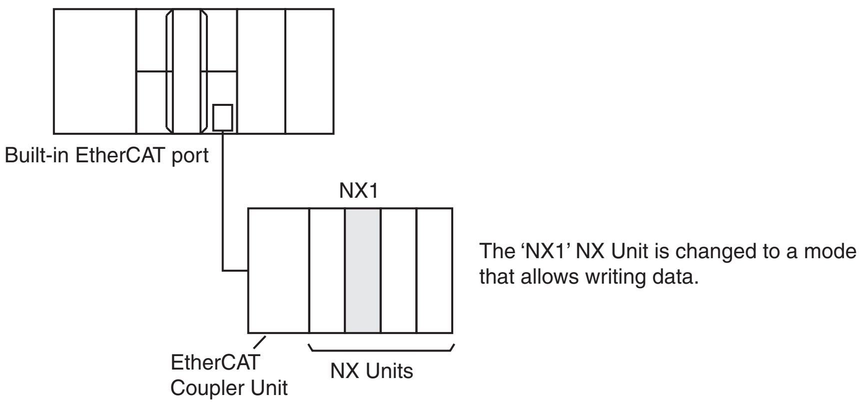
The following notation example changes the ‘NX1’ NX Unit to a mode that allows writing data. A variable that is named ‘NX1’ with a data type of _sNXUNIT_ID is assigned to the NX Unit to change.

LD



ST

```
NX_ChangeWriteMode_instance(A, NX1, abc, def,
                            ghi, jkl, mno);
```



Related System-defined Variables

Name	Meaning	Data type	Description
_EC_MBXSlavTbl[i] "i" is the node address.	Message Communications Enabled Slave Table	BOOL	This variable indicates whether communications are possible for each slave. TRUE: Communications are possible. FALSE: Communications are not possible.
_NXB_UnitMsgActiveTbl[i]	NX Unit Message Enabled Status	BOOL	This table indicates the slaves that can perform message communications. Use this variable to confirm that communications with the relevant slave are possible.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* (page 2-3) for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- If you specify a Unit that is assigned to a motion control axis (data type _sAXIS_REF) for *UnitProxy*, a Controller error will occur in the Motion Control Function Module. If that occurs, use the ResetMCError instruction (page 2-830) to reset the Controller error.
- For *UnitProxy,* specify the device variable that is assigned to the EtherCAT Coupler Unit, an NX Unit on the EtherCAT Coupler Unit, or an NX Unit connected to the NX bus of the CPU Unit in the I/O Map of the Sysmac Studio. Refer to the *Sysmac Studio Version 1 Operation Manual* (Cat. No. W504-E1-07 or later) for details on assigning device variables.
- If the NX_ChangeWriteMode instruction is executed during execution of another NX_ChangeWriteMode instruction or execution of the RestartNXUnit instruction (page 2-844), the RestartNXUnit instruction that is executed later will be queued. Up to 192 instructions can be queued. A building error will occur if an attempt is made to queue more than 192 instructions. The time that an instruction is queued is not included in the timeout time.
- The value of *Busy* is TRUE while the instruction is queued.
- This instruction is related to NX Message Communications Errors. If too many instructions that are related to NX Message Communications Errors are executed at the same time, an NX Message Communications Error will occur. Refer to *A-3 Instructions Related to NX Message Communications Errors*) for a list of the instructions that are related to NX Message Communications Errors.
- You cannot execute this instruction during execution of the following instructions: EC_DisconnectSlave, EC_ConnectSlave, EC_ChangeEnableSetting, ResetECError, RestartNXUnit, and NX_ChangeWriteMode. An error will occur if you attempt to execute it.

- *Error* is TRUE if an error occurred. The meanings of the values of *ErrorID* and *ErrorIDEx* are given in the following table.

Value of *ErrorID*	Value of *ErrorIDEx*	Meaning
16#0419	16#0000 0000	The data type of *UnitProxy* is not correct.
16#2C00	16#0000 0401	The specified Unit does not support the instruction.
	16#0000 1001 16#0000 1002 16#0017 0000 16#0020 0000 16#0021 0000	An input parameter, output parameter, or in-out parameter is incorrect. Confirm that the intended parameter is used for the input parameter, output parameter, or in-out parameter.
	16#0000 1010	The data size of the specified NX object does not agree with the data size specified in *WriteDat*.
	16#0000 1101	The correct Unit was not specified. Check the Unit.
	16#0000 110B	The size of the read data is too large. Make sure that the read data specification is correct.
	16#0000 1110	There is no object that corresponds to the value of *Obj.Index.*
	16#0000 1111	There is no object that corresponds to the value of *Obj.Subindex.*
	16#0000 2101	The specified NX object cannot be written.
	16#0000 2110	The value of *WriteDat* exceeds the range of the values of the NX object to write.
	16#0000 2210	The specified Unit is not in a mode that allows writing data.
	16#0000 2213	Instruction execution was not possible because the specified Unit was performing an I/O check. Execute the instruction after the I/O check is completed.
	16#0000 2230	The status of the specified Unit does not agree with the value of the read source or write destination NX object. Take the following actions if the value of *Obj.Index* is between 0x6000 and 0x6FFF or between 0x7000 and 0x7FFF. • Delete the read source or write designation NX object from the I/O allocation settings. • Reset the error for the specified Unit. • Place the specified Unit in a mode that does not allow writing data.
	16#0000 2231	Instruction execution was not possible because the specified Unit was performing initialization. Wait for the Unit to start normal operation and then execute the instruction.
	16#0000 250F	Hardware access failed. Execute the instruction again.
	16#0000 2601 16#0000 2602 16#0010 0000	The specified Unit does not support this instruction. Check the version of the Unit.
	16#0000 2603	Execution of the instruction failed. Execute the instruction again. Make sure that at least one channel is enabled in the selections of the channels to use.
	16#0000 2621	The NX Unit is not in a status in which it can acknowledge the instruction. Wait for a while and then execute the instruction again.
	16#0001 0000	The specified Unit does not exist. Make sure that the Unit configuration is correct.
	16#0011 0000	The specified port number does not exist. Make sure that the Unit configuration is correct.
	16#0012 0000 16#0013 0000 16#0015 0000 16#0016 0000	The value of *UnitProxy* is not correct. Set the variable that indicates the specified EtherCAT Coupler Unit again.

Value of *ErrorID*	Value of *ErrorIDEx*	Meaning
16#2C00	16#0014 0000	The specified node address is not correct. Make sure that the Unit configuration is correct.
	16#0030 0000 16#8001 0000	The specified Unit is busy. Execute the instruction again.
	16#0031 0000	The specified Unit is not supported for connection. Check the version of the Unit.
	16#80000000 16#80050000 16#81010000 16#81020000 16#82020000 16#82030000 16#82060000 to 16#8FFF0000 16#90010000 to 16#FFFE0000	An error occurred in the communications network. Execute the instruction again.
	16#8002 0000 16#8003 0000 16#8103 0000 16#8200 0000	An error occurred in the communications network. Reduce the amount of communications traffic.
	16#8004 0000 16#8100 0000 16#8201 0000 16#8204 0000 16#8205 0000 16#9000 0000	An error occurred in the communications network. Check the Unit and cable connections. Make sure that the power supply to the Unit is ON.
16#2C01	16#0000 0000	An attempt was made to queue more than 192 NX_ChangeWrite-Mode and RestartNXUnit instructions.
16#2C02	16#0000 0000	A timeout occurred during communications.
16#2C05	---	An error occurred in the EtherCAT network. Check the value of *UnitProxy* and the EtherCAT network configu-ration.
16#2C07	16#0000 0000	A slave that cannot be specified for the instruction was connected at the slave node address of the specified Unit. Check the value of *UnitProxy* and the EtherCAT network configuration.

Version Information

A CPU Unit with unit version 1.05 or later and Sysmac Studio version 1.06 or higher are required to use this instruction.

Sample Programming

Refer to the sample programming for the NX_WriteObj instruction (page 2-954).

NX_SaveParam

The NX_SaveParam instruction saves the data that was written to the specified EtherCAT Coupler Unit or NX Unit.

Instruction	Name	FB/FUN	Graphic expression	ST expression
NX_Save Param	Save NX Unit Parameters	FB	NX_SaveParam_instance NX_SaveParam Execute Done UnitProxy Busy TimeOut Error ErrorID ErrorIDEx	NX_SaveParam_instance(Execute, UnitProxy, TimeOut, Done, Busy, Error, ErrorID, ErrorIDEx);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
UnitProxy	Specified Unit	Input	Unit for which to save data	---	---	*
TimeOut	Timeout time		Timeout time If 0 is set, the timeout time is 2.0 s.	0 to 60,000	ms	2000 (2.0s)

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
UnitProxy	Refer to *Function* for details on the structure _sNXUNIT_ID.																			
TimeOut							OK													

Function

The NX_SaveParam instruction saves the data that was written to an EtherCAT Coupler Unit, an NX Unit on the EtherCAT Coupler Unit, or an NX Unit connected to the NX bus of the CPU Unit. The Unit for which to save the data is specified with *UnitProxy.*

After the completion of saving the data, the value of *Done* changes to TRUE. Use the NX_WriteObj instruction (page 2-954) to write the data. Even if power is interrupted after this instruction is executed, the values of the data with a power OFF retain attribute are retained.

TimeOut specifies the timeout time. If a response does not return within the timeout time, it is assumed that communications failed. In that case, the Unit data is not saved.

The data type of *UnitProxy* is structure _sNXUNIT_ID. The meanings of the members are as follows:

Name	Meaning	Description	Data type
UnitProxy	Specified Unit	Unit for which to save data	_sNXUNIT_ID
NodeAdr	Node address	Node address of the Communications Coupler Unit	UINT
IPAdr	IP address	IP address of the Communications Coupler Unit	BYTE[5]
UnitNo	Unit number	Unit number of specified Unit	UDINT
Path	Path	Path information to the specified Unit	BYTE[64]
PathLength	Valid path length	Valid path length	USINT

Pass the device variable that is assigned to the specified Unit to *UnitProxy*.

Related Instructions and Execution Procedure

Depending on the attributes of the data that you write to an EtherCAT Coupler Unit, an NX Unit on the EtherCAT Coupler Unit, or an NX Unit connected to the NX bus of the CPU Unit, you must execute this instruction along with other instructions. The procedures for each case are given below.

● Execution Procedure 1

Use the following procedure to write data with the following attributes.

- Power OFF Retain attribute
- The values are updated when the Unit is restarted.

1. Use the NX_ChangeWriteMode instruction (page 2-851) to change the Unit to a mode that allows writing data.
2. Use the NX_WriteObj instruction (page 2-954) to write data to the Unit.
3. Use the NX_SaveParam instruction to save the data that you wrote.
4. Use the RestartNXUnit instruction (page 2-844) to restart the Unit.

● Execution Procedure 2

Use the following procedure to write data with the following attributes.

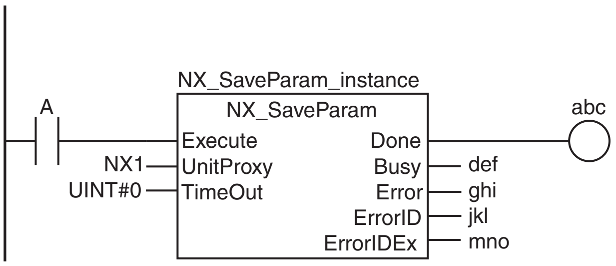
- Power OFF Retain attribute
- The values are updated as soon as they are written.

1. Use the NX_WriteObj instruction (page 2-954) to write data to the Unit.
2. Use the NX_SaveParam instruction to save the data that you wrote.

Notation Example

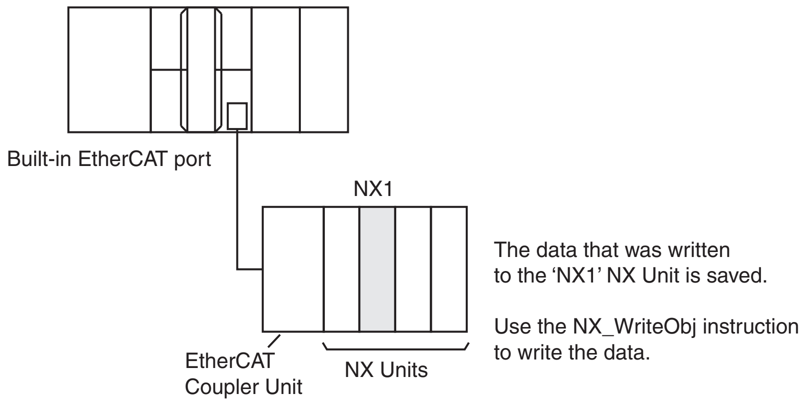
The following notation example saves the data that was written to the 'NX1' NX Unit. A variable that is named 'NX1' with a data type of _sNXUNIT_ID is assigned to the NX Unit.

LD



ST

```
NX_SaveParam_instance(A, NX1, UINT#0,
                       abc, def, ghi, jkl, mno);
```



Related System-defined Variables

Name	Meaning	Data type	Description
_EC_MBXSlavTbl[i] "i" is the node address.	Message Communications Enabled Slave Table	BOOL	This variable indicates whether communications are possible for each slave. TRUE: Communications are possible. FALSE: Communications are not possible.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* (page 2-3) for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction will not end in an error even if the Unit specified by *UnitProxy* is already saving data. The value of *Busy* remains at TRUE and the value of *Done* changes to TRUE when saving the data is completed. Requests to save data are not queued.
- An error will not occur even if this instruction is executed without writing data to the Unit.
- Some of the Units have restrictions in the number of times that you can write data. Refer to the manuals for the specific Units for details.
- For *UnitProxy,* specify the device variable that is assigned to the EtherCAT Coupler Unit, an NX Unit on the EtherCAT Coupler Unit, or an NX Unit connected to the NX bus of the CPU Unit in the I/O Map of the Sysmac Studio. Refer to the *Sysmac Studio Version 1 Operation Manual* (Cat. No. W504-E1-07 or later) for details on assigning device variables.
- To write and save data with a Power OFF Retain attribute, execute the NX_SaveParam instruction after you execute the NX_WriteObj instruction (page 2-970). If you restart the Unit before you execute the NX_SaveParam instruction, the previous NX object data is restored.
- This instruction is related to NX Message Communications Errors. If too many instructions that are related to NX Message Communications Errors are executed at the same time, an NX Message Communications Error will occur. Refer to *A-3 Instructions Related to NX Message Communications Errors* for a list of the instructions that are related to NX Message Communications Errors.

- *Error* is TRUE if an error occurred. The meanings of the values of *ErrorID* and *ErrorIDEx* are given in the following table.

Value of *ErrorID*	Value of *ErrorIDEx*	Meaning
16#0400	16#0000 0000	• The value of *UnitProxy* is outside of the valid range. • The value of *TimeOut* is outside of the valid range.
16#0419	16#0000 0000	The data type of *UnitProxy* is not correct.
16#2C00	16#0000 0401	The specified Unit does not support the instruction.
	16#0000 1001 16#0000 1002 16#0017 0000 16#0020 0000 16#0021 0000	An input parameter, output parameter, or in-out parameter is incorrect. Confirm that the intended parameter is used for the input parameter, output parameter, or in-out parameter.
	16#0000 1010	The data size of the specified NX object does not agree with the data size specified in *WriteDat*.
	16#0000 1101	The correct Unit was not specified. Check the Unit.
	16#0000 110B	The size of the read data is too large. Make sure that the read data specification is correct.
	16#0000 1110	There is no object that corresponds to the value of *Obj.Index.*
	16#0000 1111	There is no object that corresponds to the value of *Obj.Subindex.*
	16#0000 2101	The specified NX object cannot be written.
	16#0000 2110	The value of *WriteDat* exceeds the range of the values of the NX object to write.
	16#0000 2210	The specified Unit is not in a mode that allows writing data.
	16#0000 2213	Instruction execution was not possible because the specified Unit was performing an I/O check. Execute the instruction after the I/O check is completed.
	16#0000 2230	The status of the specified Unit does not agree with the value of the read source or write destination NX object. Take the following actions if the value of *Obj.Index* is between 0x6000 and 0x6FFF or between 0x7000 and 0x7FFF. • Delete the read source or write designation NX object from the I/O allocation settings. • Reset the error for the specified Unit. • Place the specified Unit in a mode that does not allow writing data.
	16#0000 2231	Instruction execution was not possible because the specified Unit was performing initialization. Wait for the Unit to start normal operation and then execute the instruction.
	16#0000 250F	Hardware access failed. Execute the instruction again.
	16#0000 2601 16#0000 2602 16#0010 0000	The specified Unit does not support this instruction. Check the version of the Unit.
	16#0000 2603	Execution of the instruction failed. Execute the instruction again. Make sure that at least one channel is enabled in the selections of the channels to use.
	16#0000 2621	The NX Unit is not in a status in which it can acknowledge the instruction. Wait for a while and then execute the instruction again.
	16#0001 0000	The specified Unit does not exist. Make sure that the Unit configuration is correct.
	16#0011 0000	The specified port number does not exist. Make sure that the Unit configuration is correct.

Value of *ErrorID*	Value of *ErrorIDEx*	Meaning
16#2C00	16#0012 0000 16#0013 0000 16#0015 0000 16#0016 0000	The value of *UnitProxy* is not correct. Set the variable that indicates the specified EtherCAT Coupler Unit again.
	16#0014 0000	The specified node address is not correct. Make sure that the Unit configuration is correct.
	16#0030 0000 16#8001 0000	The specified Unit is busy. Execute the instruction again.
	16#0031 0000	The specified Unit is not supported for connection. Check the version of the Unit.
	16#80000000 16#80050000 16#81010000 16#81020000 16#82020000 16#82030000 16#82060000 to 16#8FFF0000 16#90010000 to 16#FFFE0000	An error occurred in the communications network. Execute the instruction again.
	16#8002 0000 16#8003 0000 16#8103 0000 16#8200 0000	An error occurred in the communications network. Reduce the amount of communications traffic.
	16#8004 0000 16#8100 0000 16#8201 0000 16#8204 0000 16#8205 0000 16#9000 0000	An error occurred in the communications network. Check the Unit and cable connections. Make sure that the power supply to the Unit is ON.
16#2C01	16#0000 0000	The number of instructions that can be simultaneously executed was exceeded.
16#2C02	16#0000 0000	A timeout occurred during communications.

Version Information

A CPU Unit with unit version 1.05 or later and Sysmac Studio version 1.06 or higher are required to use this instruction.

Sample Programming

Refer to the sample programming for the NX_WriteObj instruction (page 2-954).

NX_ReadTotalPowerOnTime

The NX_ReadTotalPowerOnTime instruction reads the total power ON time from a Communications Coupler Unit or NX Unit.

Instruction	Name	FB/FUN	Graphic expression	ST expression
NX_Read TotalPower OnTime	Read NX Unit Total Power ON Time	FB	NX_ReadTotalPowerOnTime_instance NX_ReadTotalPowerOnTime Execute — Done UnitProxy — Busy Error ErrorID ErrorIDEx TotalPowerOnTime	NX_ReadTotalPowerONTime_instance(Execute, UnitProxy, Done, Busy, Error, ErrorID, ErrorIDEx, TotalPowerOnTime);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
UnitProxy	Specified Unit	Input	Specifies the target NX Unit.	---	---	*
TotalPow-erOnTime	Total power ON time	Output	Stores the total power ON time that was read.	Depends on data type.		0

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
UnitProxy	Refer to *Function* for details on the structure _sNXUNIT_ID.																			
TotalPow-erOnTime																OK				

Function

The NX_ReadTotalPowerOnTime instruction reads the approximate total power ON time from a Communications Coupler Unit, an NX Unit on the Communications Coupler Unit, or an NX Unit connected to the NX bus of the CPU Unit.
The accuracy is 1 hour per month.

The Unit from which the total power ON time is read is specified with *UnitProxy*.
When the value of *Done* changes to TRUE, the total power ON time has been read.

The data type of *UnitProxy* is structure _sNXUNIT_ID. The meanings of the members are as follows:

Name	Meaning	Description	Data type
UnitProxy	Specified Unit	Specified Unit	_sNXUNIT_ID
NodeAdr	Node address	Node address of the Communications Coupler Unit	UINT
IPAdr	IP address	IP address of the Communications Coupler Unit	BYTE[5]
UnitNo	Unit number	Unit number of specified NX Unit	UDINT
Path	Path	Path information to the specified Unit	BYTE[64]
PathLength	Valid path length	Valid path length	USINT

Pass a device variable that is assigned to the specified Communications Coupler Unit, an NX Unit on the Communications Coupler Unit, or an NX Unit connected to the NX bus of the CPU Unit to *Unit-Proxy*.

Version Combinations

There are combinations in which you can read the total power ON time depending on the version of the Communications Coupler Unit connected to the CPU Unit, NX Unit on the Communications Coupler Unit, or NX Unit connected to the NX bus of the CPU Unit.

● EtherCAT Slave Terminal

Unit	Version of NX Unit	Version of EtherCAT Coupler Unit
Digital I/O Unit	Version 1.0 or later	Version 1.2 or later
Analog I/O Unit		
System Unit		
Position Interface Unit	Version 1.1 or later	
Temperature Input Unit		

● NX Unit on NX1P2 CPU Unit

Unit	Version of NX Unit
Digital I/O Unit	Version 1.0 or later
Analog I/O Unit	
System Unit	
Position Interface Unit	Version 1.1 or later
Temperature Input Unit	

Related System-defined Variables

Name	Meaning	Data type	Description
_EC_MBXSlavTbl[i] "i" is the node address.	Message Communications Enabled Slave Table	BOOL	This variable indicates when communications are possible for each slave. TRUE: Communications are possible. FALSE: Communications are not possible.
_NXB_UnitMsgActiveTbl[i]	NX Unit Message Enabled Status	BOOL	This table indicates the slaves that can perform message communications. Use this variable to confirm that communications with the relevant slave are possible.

Additional Information

If this instruction is executed by the Simulator, *Busy* changes to TRUE for only one task period after *Execute* changes from FALSE to TRUE.

Busy changes to FALSE and *Done* changes to TRUE the next task period.

The value that is read in *TotalPowerOnTime* will be 0.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal end of processing.
- Refer to *Using this Section* (page 2-3) for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- For *UnitProxy*, specify the device variable that is assigned to the EtherCAT Coupler Unit, an NX Unit on the EtherCAT Coupler Unit, or an NX Unit connected to the NX bus of the CPU Unit in the I/O Map of the Sysmac Studio. Refer to the *Sysmac Studio Version 1 Operation Manual* (Cat. No. W504-E1-07 or later) for details on assigning device variables.
  An error will occur if you specify an NX-series CPU Unit for *UnitProxy*.
- There are restrictions in the number of Units that depend on the Communications Coupler Unit. Refer to the manual for your Communications Coupler Unit for details.
- *Error* is TRUE if an error occurred. The meanings of the values of *ErrorID* and *ErrorIDEx* are given in the following table.

Value of *ErrorID*	Value of *ErrorIDEx*	Meaning
16#0400	16#0000 0000	The value of *UnitProxy* is outside of the valid range.
16#0419	16#0000 0000	The data type of *UnitProxy* is not correct.

Value of *ErrorID*	Value of *ErrorIDEx*	Meaning
16#2C00	16#0000 0401	The specified Unit does not support the instruction.
	16#0000 1001 16#0000 1002 16#0017 0000 16#0020 0000 16#0021 0000	An input parameter, output parameter, or in-out parameter is incorrect. Confirm that the intended parameter is used for the input parameter, output parameter, or in-out parameter.
	16#0000 1010	The data size of the specified NX object does not agree with the data size specified in *WriteDat*.
	16#0000 1101	The Unit is not correct. Check the Unit.
	16#0000 110B	The size of the read data is too large. Make sure that the read data specification is correct.
	16#0000 1110	There is no object that corresponds to the value of *Obj.Index*.
	16#0000 1111	There is no object that corresponds to the value of *Obj.Subindex*.
	16#0000 2101	The specified NX object cannot be written.
	16#0000 2110	The value of *WriteDat* exceeds the range of the values of the NX object to write.
	16#0000 2210	The specified Unit is not in a mode that allows writing data.
	16#0000 2213	Instruction execution was not possible because the specified Unit was performing an I/O check. Execute the instruction after the I/O check is completed.

Value of *ErrorID*	Value of *ErrorIDEx*	Meaning
16#2C00	16#0000 2230	The status of the specified Unit does not agree with the value of the read source or write destination NX object. Take the following actions if the value of *Obj.Index* is between 0x6000 and 0x6FFF or between 0x7000 and 0x7FFF. • Delete the read source or write destination NX object from the I/O allocation settings. • Reset the error for the specified Unit. • Place the specified Unit in a mode that does not allow writing data.
	16#0000 2231	Instruction execution was not possible because the specified Unit was performing initialization. Wait for the Unit to start normal operation and then execute the instruction.
	16#0000 250F	Hardware access failed. Execute the instruction again.
	16#0000 2601 16#0000 2602 16#0010 0000	The specified Unit does not support this instruction. Check the version of the Unit.
	16#0000 2603	Execution of the instruction failed. Execute the instruction again. Make sure that at least one channel is enabled in the selections of the channels to use.
	16#0000 2621	The NX Unit is not in a status in which it can acknowledge the instruction. Wait for a while and then execute the instruction again.
	16#0001 0000	The specified Unit does not exist. Make sure that the Unit configuration is correct.
	16#0011 0000	The specified port number does not exist. Make sure that the Unit configuration is correct.
	16#0012 0000 16#0013 0000 16#0015 0000 16#0016 0000	The value of *UnitProxy* is not correct. Set the variable that indicates the specified EtherCAT Coupler Unit again.
	16#0014 0000	The specified node address is not correct. Make sure that the Unit configuration is correct.
	16#0030 0000 16#8001 0000	The specified Unit is busy. Execute the instruction again.
	16#0031 0000	The specified Unit is not supported for connection. Check the version of the Unit.
	16#8000 0000 16#8005 0000 16#8101 0000 16#8102 0000 16#8202 0000 16#8203 0000 16#8206 0000 to 16#8FFF 0000 16#9001 0000 to 16#FFFE 0000	An error occurred in the communications network. Execute the instruction again.
	16#8002 0000 16#8003 0000 16#8103 0000 16#8200 0000	An error occurred in the communications network. Reduce the amount of communications traffic.

Value of *ErrorID*	Value of *ErrorIDEx*	Meaning
16#2C00	16#8004 0000 16#8100 0000 16#8201 0000 16#8204 0000 16#8205 0000 16#9000 0000	An error occurred in the communications network. Check the Unit and cable connections. Make sure that the power supply to the Unit is ON.
16#2C01	16#0000 0000	The number of instructions that can be simultaneously executed was exceeded.
16#2C02	16#0000 0000	A timeout occurred during communications.
16#2C05	---	An error occurred in the EtherCAT network. Check the value of *UnitProxy* and the EtherCAT network configuration.
16#2C07	16#0000 0000	A slave that cannot be specified for the instruction was connected at the slave node address of the specified Unit. Check the value of *UnitProxy* and the EtherCAT network configuration.
16#2C08	16#00000000	The total power ON time could not be read.

Sample Programming

Two modes are created in a program: maintenance mode and run mode.

With this sample, if the button to read the total power ON time is pressed while in maintenance mode, the total power ON time of Unit 3 (set in advance) is read.

If the total power ON time exceeds 5 years, a lamp is lit to indicate that the Unit replacement is necessary.

If the button for completion of Unit replacement is pressed after replacing the Unit, the Unit replacement warning lamp turns OFF.

The following system configuration is used.

Unit	Description
Unit 1	NX Unit (ID)
Unit 2	NX Unit (OD)
Unit 3	NX Unit (Unit from which to read the total power ON time)

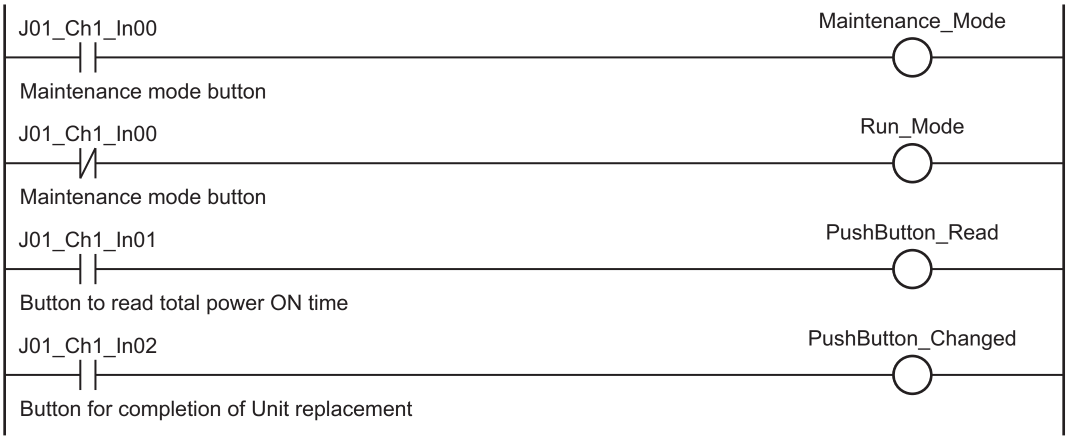
Definitions of Variables

LD

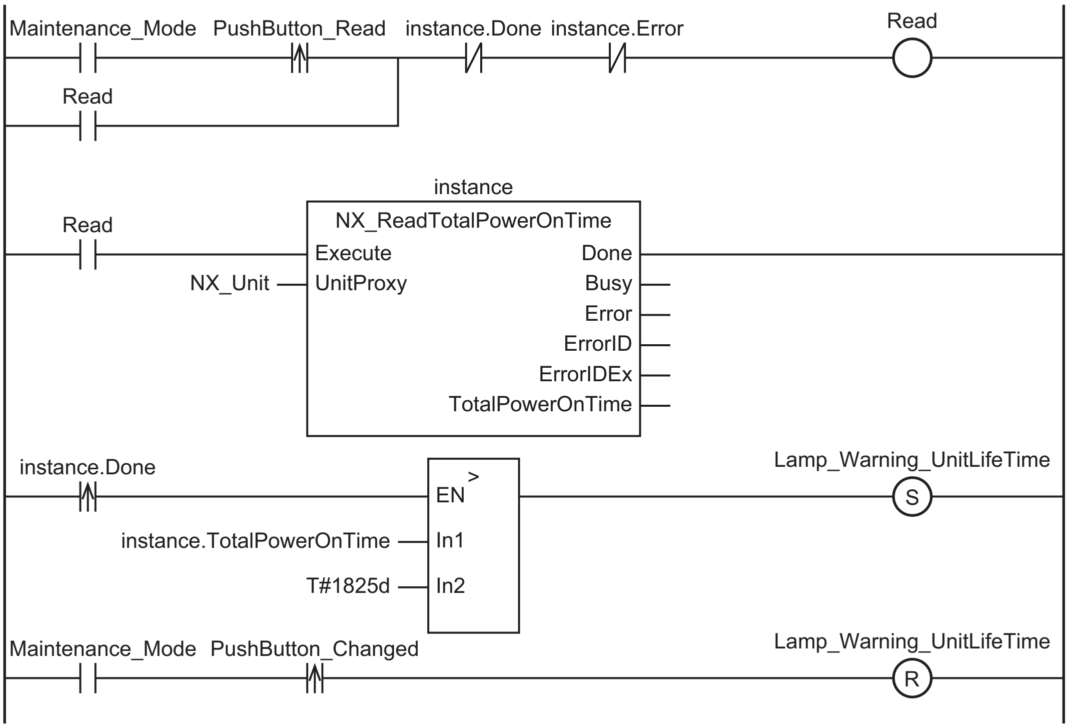
Internal Variables	Variable	Data type	Initial value	Comment
	Maintenance_Mode	BOOL	FALSE	Maintenance mode
	Run_Mode	BOOL	FALSE	Run mode
	PushButton_Read	BOOL	FALSE	Reading the total power ON time
	PushButton_Changed	BOOL	FALSE	Completion of Unit replacement
	Lamp_Warning_UnitLifeTime	BOOL	FALSE	Unit replacement warning
	Read	BOOL	FALSE	
	instance	NX_ReadTotal-PowerOnTime		

External Variables	Variable	Data type	Comment
	NX_Unit	_sNXUNIT_ID	
	J01_Ch1_In00	BOOL	Maintenance mode button
	J01_Ch1_In01	BOOL	Button to read total power ON time
	J01_Ch1_In02	BOOL	Button for completion of Unit replacement
	J02_Ch1_Out00	BOOL	Unit replacement warning lamp

Get button status.



Read total power ON time.



Output warning to lamp.

ST

Internal Variables	Variable	Data type	Initial value	Comment
	Maintenance_Mode	BOOL	FALSE	Maintenance mode
	Run_Mode	BOOL	FALSE	Run mode
	PushButton_Read	BOOL	FALSE	Reading the total power ON time
	PushButton_Changed	BOOL	FALSE	Completion of Unit replacement
	Lamp_Warning_UnitLifeTime	BOOL	FALSE	Unit replacement warning
	Read	BOOL	FALSE	
	instance	NX_ReadTotalPowerOn-Time		
	RS_instance	RS		
	RS_instance2	RS		
	R_TRIG_instance1	R_TRIG		
	R_TRIG_instance2	R_TRIG		
	R_TRIG_instance3	R_TRIG		
	PushButton_Read_R_TRIG	BOOL		
	instance_Done_R_TRIG	BOOL		
	PushButton_Change_R_TRIG	BOOL		

External Variables	Variable	Data type	Comment
	NX_Unit	_sNXUNIT_ID	
	J01_Ch1_In00	BOOL	Maintenance mode button
	J01_Ch1_In01	BOOL	Button to read total power ON time
	J01_Ch1_In02	BOOL	Button for completion of Unit replacement
	J02_Ch1_Out00	BOOL	Unit replacement warning lamp

```
// Get button status.
Maintenance_Mode   := J01_Ch1_In00;
Run_Mode           := NOT(J01_Ch1_In00);
PushButton_Read    := J01_Ch1_In01;
PushButton_Changed := J01_Ch1_In02;

R_TRIG_instance1(Clk:= PushButton_Read, Q=>PushButton_Read_R_TRIG);

// Read total power ON time.
Rs_instance( Set:= (Maintenance_Mode & PushButton_Read_R_TRIG),
             Reset1:=((instance.Done) OR (instance.Error)),
             Q1=>Read);
instance(Execute:=Read, UnitProxy:=NX_Unit);

R_TRIG_instance2(Clk:= instance.Done, Q=>instance_Done_R_TRIG);
R_TRIG_instance3(Clk:= PushButton_Changed, Q=>PushButton_Changed_R_TRIG);

RS_instance2(Set:=(instance_Done_R_TRIG & (instance.TotalPowerOnTime>T#1825d)),
              Reset1:=(Maintenance_Mode & PushButton_Changed_R_TRIG),
              Q1=>Lamp_Warning_UnitLifeTime);
// Output warning to lamp.
J02_Ch1_Out00  := Lamp_Warning_UnitLifeTime;
```

Program Control Instructions

PrgStart

The PrgStart instruction enables the execution of the specified program.

Instruction	Name	FB/FUN	Graphic expression	ST expression
PrgStart	Enable Program	FUN		Out:=PrgStart(PrgName, isFirstRun);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
PrgName	Program name	Input	Name of specified program	128 bytes max. (127 single-byte alphanumeric characters plus the final NULL character)	---	*1
isFirstRun	First Program Period Flag enable	Input	Operation of the *P_First_Run* system-defined variable in the first task period when the program is executed TRUE: Change to TRUE. FALSE: Change to FALSE.	Depends on data type.	---	TRUE
Out	Normal end flag	Output	Normal end flag TRUE: Normal end FALSE: Error end	Depends on data type.	---	---

*1 If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
PrgName																				OK
isFirstRun	OK																			
Out	OK																			

Function

The PrgStart instruction enables the execution of the program specified with *PrgName*. The specified program is executed the next time the timing for executing the program occurs. An error does not occur even if the specified program is already enabled.

The specified program can be in the same task as this instruction, or it can be in a different task.

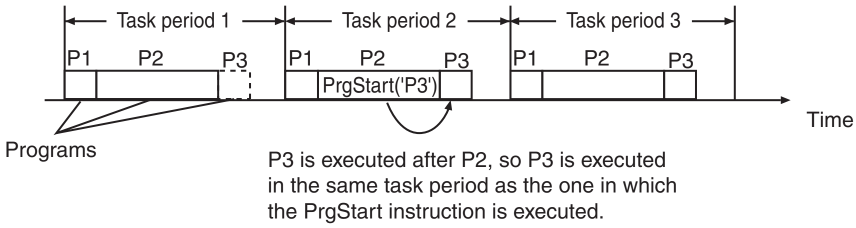
The value of *Out* is TRUE if the instruction ends normally and FALSE if the instruction ends in an error.

Operation Example When a Program in the Current Task Is Specified

An operation example is provided below for when a program is specified that is in the same task as the task that executes the instruction.

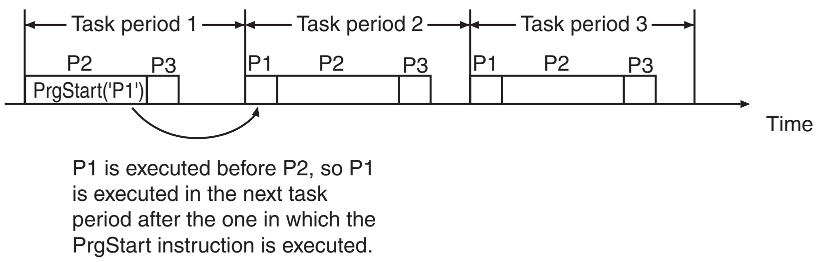
● Enabling a Program Executed After the PrgStart Instruction

- In this example, there are three programs, P1, P2, and P3, in the same task.
- P3 is disabled from task period 1.
- The PrgStart instruction with P3 specified is executed in P2 of task period 2.
- P3 is executed after P2, so P3 is executed in task period 2.
- Thereafter, P3 remains enabled even if you do not execute the PrgStart instruction with P3 specified.



● Enabling a Program Executed Before the PrgStart Instruction

- In this example, there are three programs, P1, P2, and P3, in the same task.
- P1 is disabled from task period 1.
- The PrgStart instruction with P1 specified is executed in P2 of task period 1.
- P1 is executed before P2, so P1 is executed in task period 2.
- Thereafter, P1 remains enabled even if you do not execute the PrgStart instruction with P1 specified.

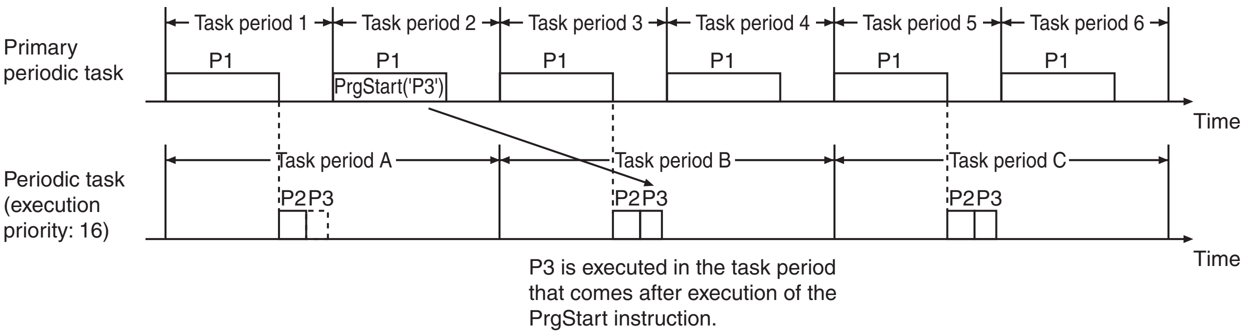


Operation Example When a Program in a Different Task Is Specified

An operation example is provided below for when a program is specified that is in a different task from the task that executes the instruction.

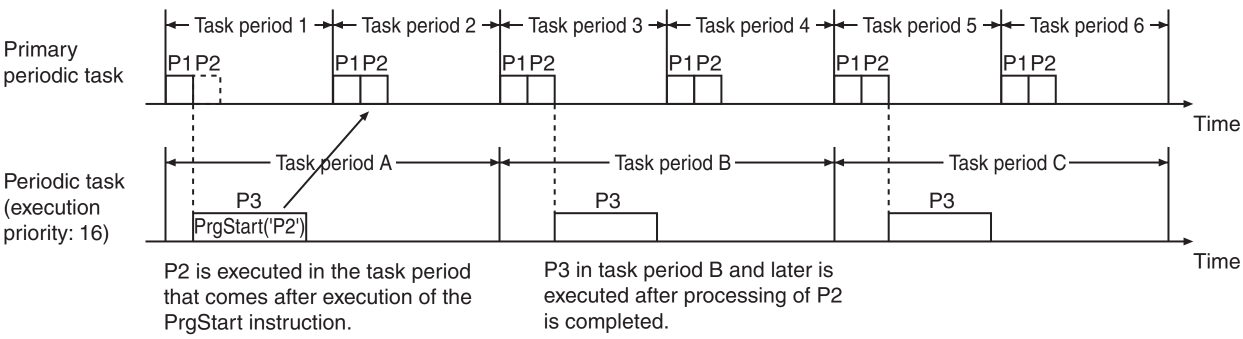
● Enabling a Program in a Task with a Lower Execution Priority Than the Current Task

- There are three programs in this example. P1 is in the primary periodic task, and P2 and P3 are in a periodic task.
- P3 is disabled from task period A of the periodic task.
- The PrgStart instruction with P3 specified is executed in P1 of task period 2 of the primary periodic task.
- P3 is executed in task period B of the periodic task, which is executed after the PrgStart instruction is executed.
- Thereafter, P3 remains enabled even if you do not execute the PrgStart instruction with P3 specified.



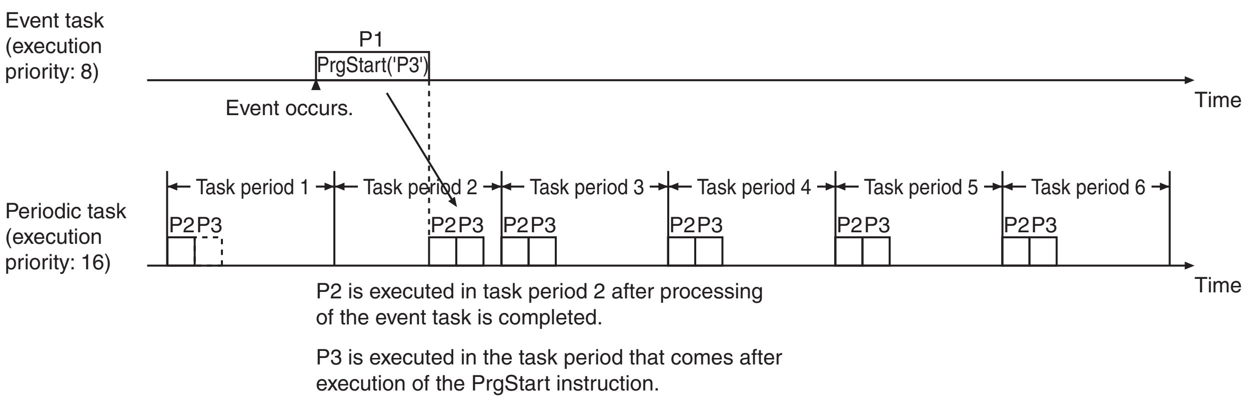
● Enabling a Program in a Task with a Higher Execution Priority Than the Current Task

- There are three programs in this example. P1 and P2 are in the primary periodic task, and P3 is in a periodic task.
- P2 is disabled from task period 1 of the primary periodic task.
- The PrgStart instruction with P2 specified is executed in P3 of task period A of the periodic task.
- P2 is executed in task period 2 of the primary periodic task, which is executed after the PrgStart instruction is executed.
- Thereafter, P2 remains enabled even if you do not execute the PrgStart instruction with P2 specified.
- The primary periodic task has a higher execution priority than a periodic task, so P3 in task period B and later is executed after processing of P2 is completed.



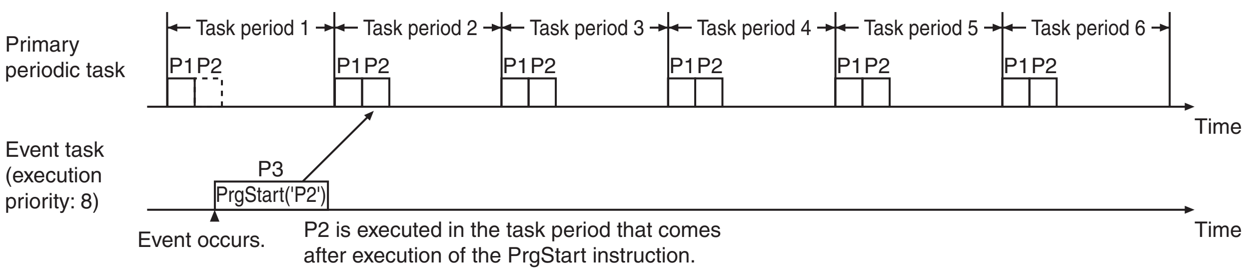
Enabling a Program in a Task with a Lower Execution Priority from an Event Task

- There are three programs in this example. P1 is in an event task (execution priority: 8), and P2 and P3 are in a periodic task (execution priority: 16).
- P3 is disabled from task period 1 of the periodic task.
- The PrgStart instruction with P3 specified is executed in the event task.
- When the event task is executed, P2 and P3 in task period 2 of the periodic task are executed after processing of the event task is completed.
- As a result, P3 in task period 2 of the periodic task is executed because it comes after execution of the PrgStart instruction.
- Thereafter, P3 remains enabled even if you do not execute the PrgStart instruction with P3 specified.



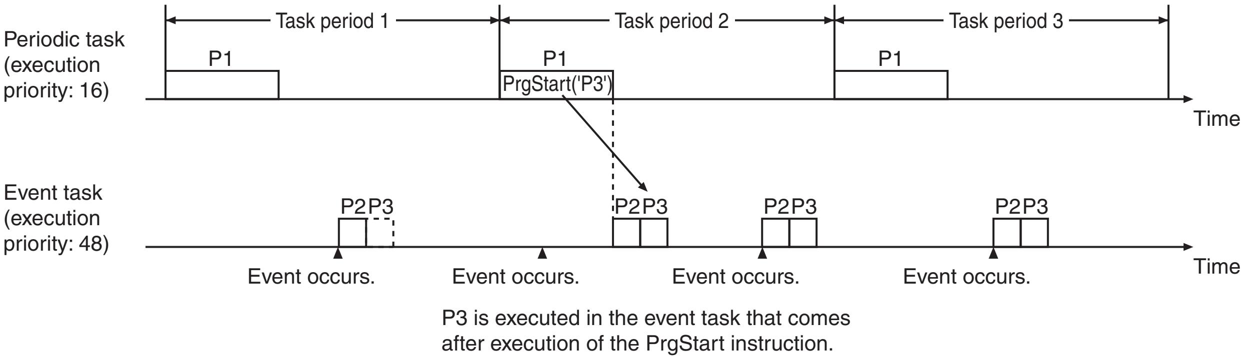
Enabling a Program in a Task with a Higher Execution Priority from an Event Task

- There are three programs in this example. P1 and P2 are in the primary periodic task, and P3 is in an event task.
- P2 is disabled from task period 1 of the primary periodic task.
- The PrgStart instruction with P2 specified is executed in the event task.
- P2 is executed in task period 2 of the primary periodic task, which is executed after the PrgStart instruction is executed.
- Thereafter, P2 remains enabled even if you do not execute the PrgStart instruction with P2 specified.



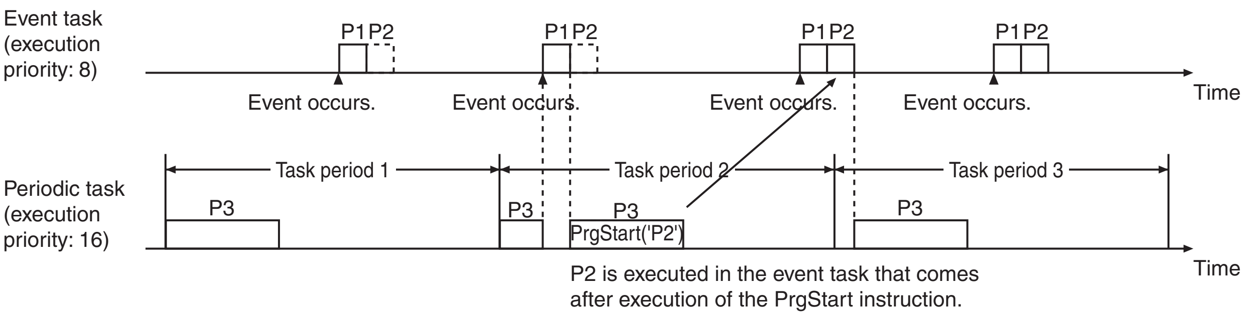
- **Enabling a Program in an Event Task with a Lower Execution Priority from a Periodic Task**

  - There are three programs in this example. P1 is in a periodic task (execution priority: 16), and P2 and P3 are in an event task (execution priority: 48).
  - P3 in the event task is disabled.
  - The PrgStart instruction with P3 specified is executed in the periodic task.
  - P3 is executed in the event task that is executed after the PrgStart instruction is executed.
  - Thereafter, P3 remains enabled even if you do not execute the PrgStart instruction with P3 specified.



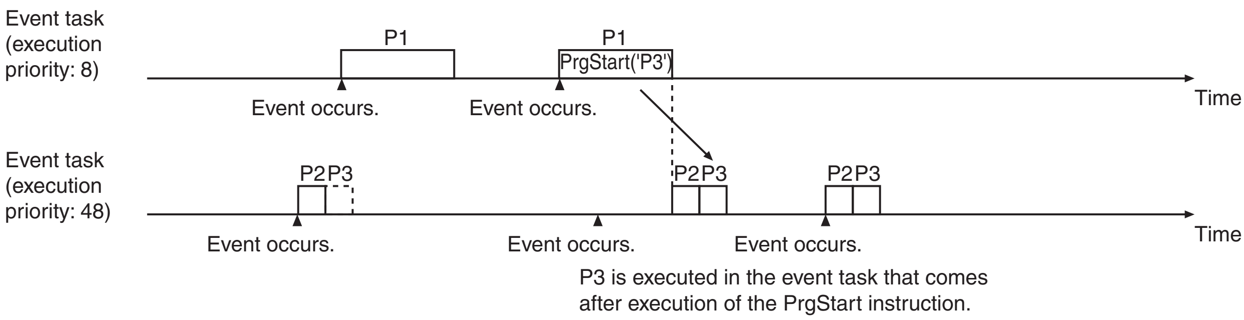
- **Enabling a Program in an Event Task with a Higher Execution Priority from a Periodic Task**

  - There are three programs in this example. P1 and P2 are in an event task (execution priority: 8), and P2 is in a periodic task (execution priority: 16).
  - P2 in the event task is disabled.
  - The PrgStart instruction with P2 specified is executed in the periodic task.
  - P2 is executed in the event task that is executed after the PrgStart instruction is executed.
  - Thereafter, P2 remains enabled even if you do not execute the PrgStart instruction with P2 specified.



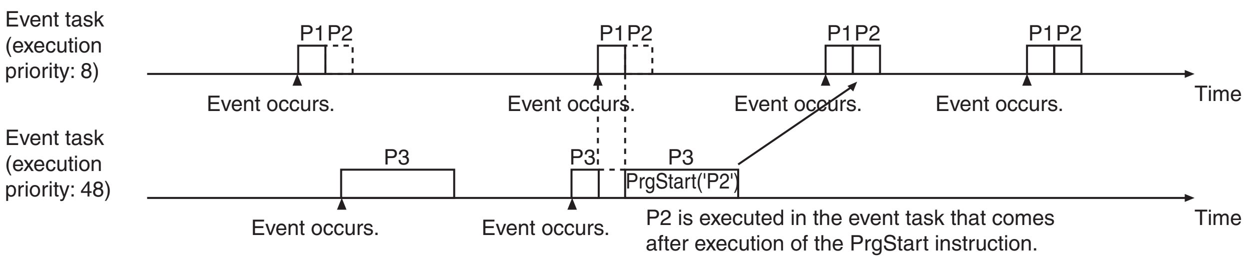
● Enabling a Program in an Event Task with a Lower Execution Priority from an Event Task

- There are three programs in this example. P1 is in an event task (execution priority: 8), and P2 and P3 are in an event task (execution priority: 48).
- P3 in the event task (execution priority: 48) is disabled.
- The PrgStart instruction with P3 specified is executed in the event task (execution priority: 8).
- P3 is executed in the event task (execution priority: 48) that is executed after the PrgStart instruction is executed.
- Thereafter, P3 remains enabled even if you do not execute the PrgStart instruction with P3 specified.



● Enabling a Program in an Event Task with a Higher Execution Priority from an Event Task

- There are three programs in this example. P1 and P2 are in an event task (execution priority: 8), and P3 is in an event task (execution priority: 48).
- P2 in the event task (execution priority: 8) is disabled.
- The PrgStart instruction with P2 specified is executed in the event task (execution priority: 48).
- P2 is executed in the event task (execution priority: 8) that is executed after the PrgStart instruction is executed.
- Thereafter, P2 remains enabled even if you do not execute the PrgStart instruction with P2 specified.



First Program Period Flag Enable (*isFirstRun*)

isFirstRun determines whether the *P_First_Run* system-defined variable is enabled as shown in the following table. If the value of *isFirstRun* is TRUE when the instruction is executed, the value of *P_First_Run* is TRUE for one task period when program execution starts. If the value of *isFirstRun* is FALSE when the instruction is executed, the value of *P_First_Run* remains FALSE even when program execution starts.

Use *isFirstRun* to perform specific processing only if specific conditions are met when program execution starts. When the specific conditions are met, change the value of *isFirstRun* to TRUE before you execute the instruction. With this program, an algorithm is used to perform specific processing when the value of *P_First_Run* is TRUE.

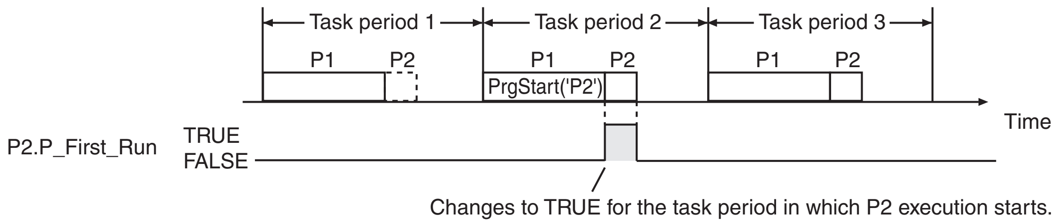
The relation between *isFirstRun* and *P_First_Run* is shown in the following table. The behavior of *P_First_Run* depends on whether the specified program is disabled or already enabled.

Value of *isFirstRun*	Status of the program	Value of *P_First_Run*
TRUE	Disabled.	Changes to TRUE for one task period when the program is executed. Changes to FALSE in the following task period.
	Already enabled.	Remains FALSE.
FALSE	---	Remains FALSE.

The following figures show examples of the relation between *isFirstRun* and *P_First_Run.*

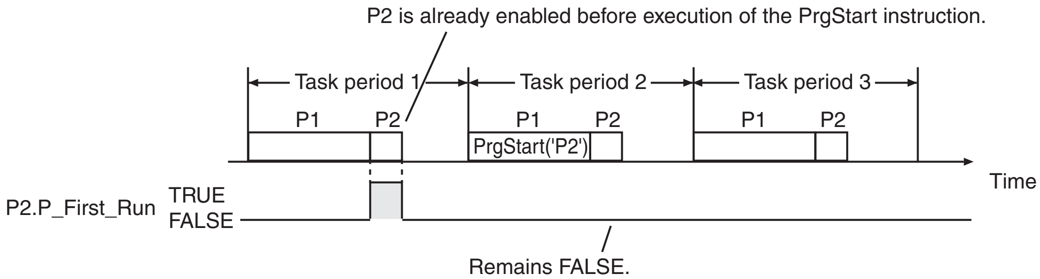
● When the Value of *isFirstRun* Is TRUE and the Program Is Disabled

The value of *P_First_Run* changes to TRUE for one task period when execution of the program starts. Then, the value of *P_First_Run* changes to FALSE.



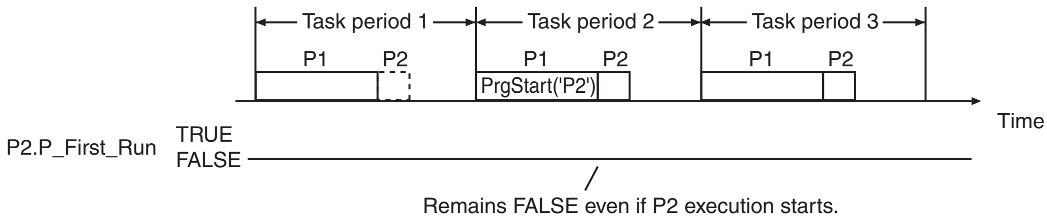
● When the Value of *isFirstRun* Is TRUE and the Program Is Already Enabled

The value of *P_First_Run* remains FALSE even if the PrgStart instruction is executed.



● When the Value of *isFirstRun* Is FALSE

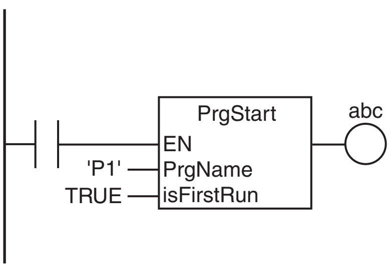
The value of *P_First_Run* remains FALSE even when execution of the program starts.



Notation Example

The following example shows the notation for specifying enabling program P1.

LD



ST

```
abc:=PrgStart('P1', TRUE);
```

Related System-defined Variables

Name	Meaning	Data type	Description
P_First_Run	First Program Period Flag	BOOL	This flag is TRUE for one task period after execution of the program starts. Otherwise it is FALSE. However, if the value of *isFirstRun* is changed to FALSE and the PrgStart instruction is executed, *P_First_Run* remains FALSE even through execution of the program starts. Use this flag to perform specific processing when execution of a program starts.
P_First_RunMode	First RUN Period Flag	BOOL	This flag is TRUE for only one task period after the operating mode of the CPU Unit is changed from PROGRAM mode to RUN mode if execution of the program is in progress. This flag remains FALSE if execution of the program is not in progress. Use this flag to perform initialization when the CPU Unit begins operation.

Additional Information

- Use the PrgStop instruction (page 2-881) to disable a specified program from the user program.
- Use the PrgStatus instruction (page 2-901) to read the status of a specified program from the user program.

Precautions for Correct Use

- An error will not occur even if you specify a program that is already in an enabled state and execute this instruction.
- If you execute this instruction more than once for the same program, the *isFirstRun* specification in the instruction instance that was executed first is used.
- If the PrgStop instruction is executed after executing the PrgStart instruction for the same program and it is executed before the program is actually executed, the program is not executed.
- If the PrgStart instruction is executed after executing the PrgStop instruction for the same program and it is executed before the execution timing for the program, the program is not disabled.
- The operation of the programs immediately after the operating mode of the CPU Unit changes to RUN mode is controlled by the setting of the *Initial Status* for each program on the Sysmac Studio. In other words, the results of executing the PrgStart or PrgStop instruction before changing to RUN mode are not valid.
- If this instruction is executed for a program in a different task, the execution timing of the specified program will depend on the task execution priority of both tasks. In some cases, the Controller may perform unexpected operation. You can execute this instruction in the first program in the task to which the specified program is assigned to make sure that the specified program is executed in the same task period as the instruction.
- The values of the internal variables, input variables, output variables, and in-out variables from the previous time that the specified program was executed are retained. To initialize these variables before you execute the program, change the value of *isFirstRun* to TRUE before you execute the instruction and then perform initialization processing in the specified program when the value of *P_First_Run* is TRUE.
- An error will occur in the following case. *Out* will be FALSE.
  - The program specified by *PrgName* does not exist.

Version Information

A CPU Unit with unit version 1.08 or later and Sysmac Studio version 1.09 or higher are required to use this instruction.

Sample Programming

Refer to the sample programming for the PrgStop instruction (page 2-881).

PrgStop

The PrgStop instruction disables execution of the specified program.

Instruction	Name	FB/FUN	Graphic expression	ST expression
PrgStop	Disable Program	FUN	(@)PrgStop EN — Out PrgName	Out:=PrgStop(PrgName);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
PrgName	Program name	Input	Name of specified program	128 bytes max. (127 single-byte alphanumeric characters plus the final NULL character)	---	*1
Out	Normal end flag	Output	Normal end flag TRUE: Normal end FALSE: Error end	Depends on data type.	---	---

*1 If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
PrgName																				OK
Out	OK																			

Function

The PrgStop instruction disables execution of the program specified with *PrgName*. The specified program is disabled from the next time the timing for executing the program occurs. An error does not occur even if the specified program is already disabled.

The specified program can be in the same task as this instruction, or it can be in a different task.

You can specify the program that contains this instruction. If you specify the program that contains the instruction, the program is executed to the end in the task period in which the instruction is executed and then the program is disabled from the next task period.

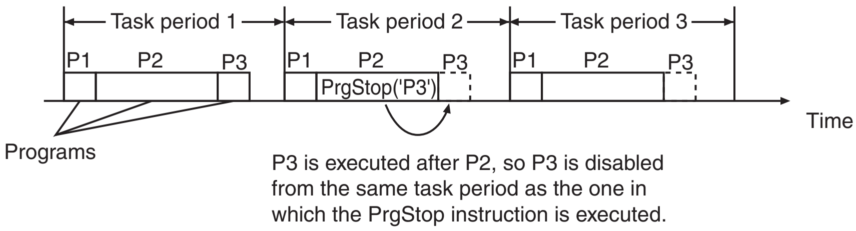
The value of *Out* is TRUE if the instruction ends normally and FALSE if the instruction ends in an error.

Operation Example When a Program in the Current Task Is Specified

An operation example is provided below for when a program is specified that is in the same task as the task that executes the instruction.

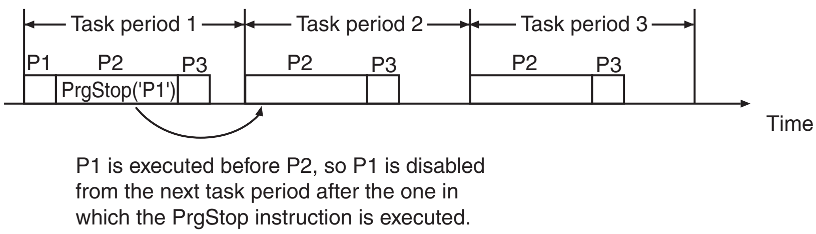
Disabling a Program Executed After the PrgStop Instruction

- In this example, there are three programs, P1, P2, and P3, in the same task.
- P3 is executed in task period 1.
- The PrgStop instruction with P3 specified is executed in P2 of task period 2.
- P3 is executed after P2, so P3 is disabled from task period 2.
- Thereafter, P3 remains disabled even if you do not execute the PrgStop instruction with P3 specified.



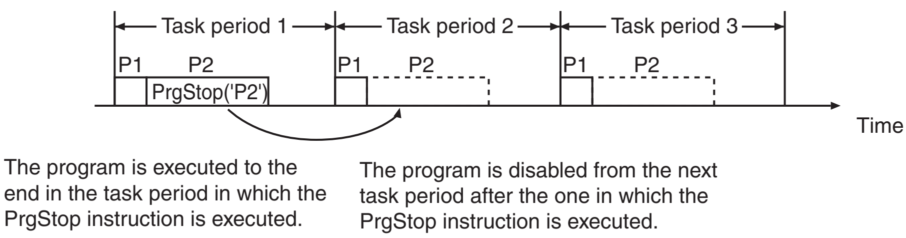
Disabling a Program Executed Before the PrgStop Instruction

- In this example, there are three programs, P1, P2, and P3, in the same task.
- P1 is executed in task period 1.
- The PrgStop instruction with P2 specified is executed in P2 of task period 1.
- P1 is executed before P2, so P1 is disabled from task period 2.
- Thereafter, P1 remains disabled even if you do not execute the PrgStop instruction with P1 specified.



Disabling the Program That Includes the PrgStop Instruction

- In this example, there are two programs, P1 and P2, in the same task.
- P2 is executed in task period 1.
- The PrgStop instruction with P2 specified is executed in P2 of task period 1.
- P2 is executed to the end of the program in task period 1.
- P2 is disabled from task period 2.
- Thereafter, P2 remains disabled even if you do not execute the PrgStop instruction with P2 specified.

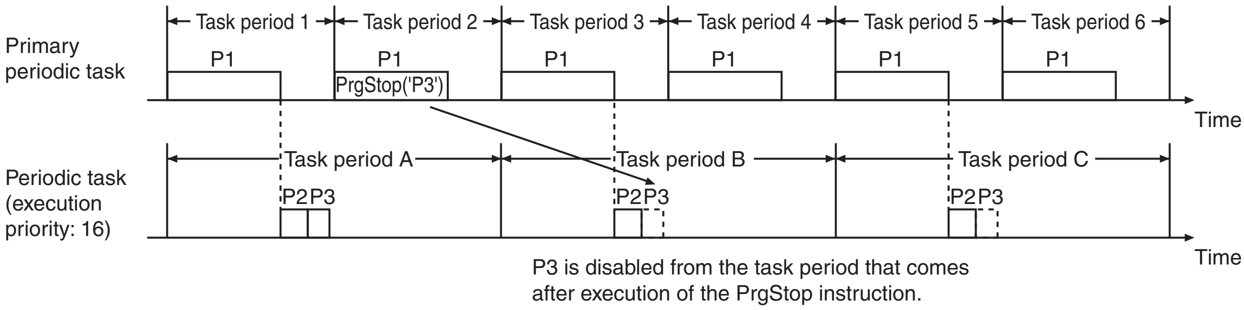


Operation Example When a Program in a Different Task Is Specified

An operation example is provided below for when a program is specified that is in a different task from the task that executes the instruction.

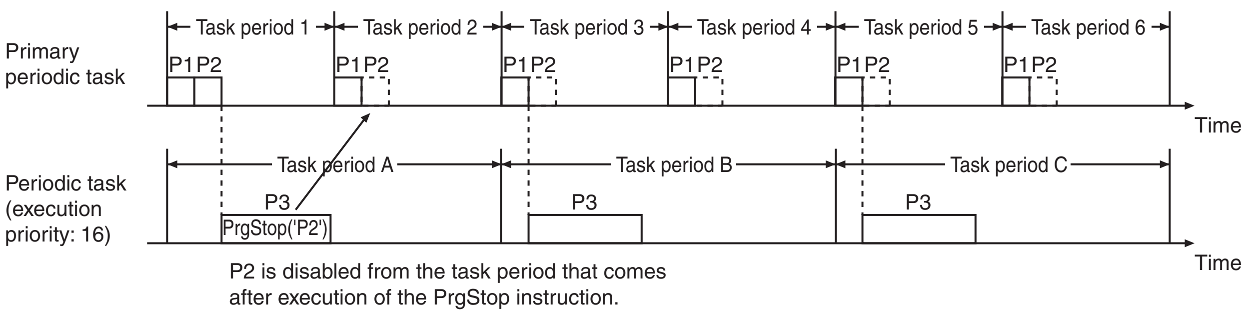
● Disabling a Program in a Task with a Lower Execution Priority Than the Current Task

- There are three programs in this example. P1 is in the primary periodic task, and P2 and P3 are in a periodic task.
- P3 is executed in task period A of the periodic task.
- The PrgStop instruction with P3 specified is executed in P1 of task period 2 of the primary periodic task.
- P3 is disabled from task period B of the periodic task, which is executed after the PrgStop instruction is executed.
- Thereafter, P3 remains disabled even if you do not execute the PrgStop instruction with P3 specified.



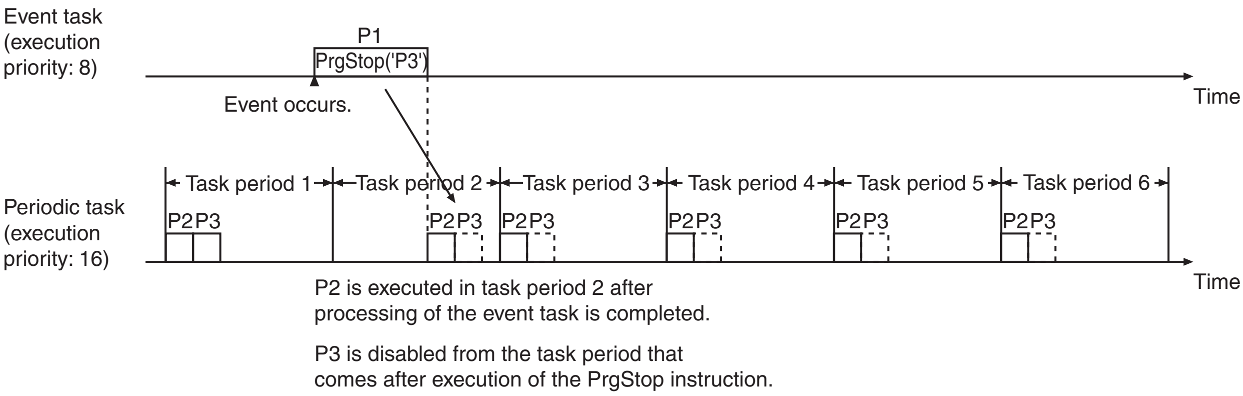
● Disabling a Program in a Task with a Higher Execution Priority Than the Current Task

- There are three programs in this example. P1 and P2 are in the primary periodic task, and P3 is in a periodic task.
- P2 is executed in task period 1 of the primary periodic task.
- The PrgStop instruction with P2 specified is executed in P3 of task period A of the periodic task.
- P2 is disabled from task period 2 of the primary periodic task, which is executed after the PrgStop instruction is executed.
- Thereafter, P2 remains disabled even if you do not execute the PrgStop instruction with P2 specified.



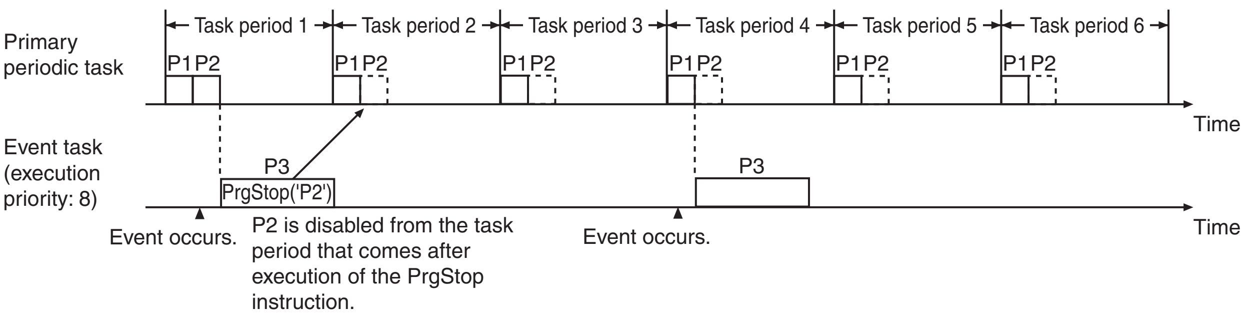
Disabling a Program in a Task with a Lower Execution Priority from an Event Task

- There are three programs in this example. P1 is in an event task (execution priority: 8), and P2 and P3 are in a periodic task (execution priority: 16).
- P3 is executed in task period 1 of the periodic task.
- The PrgStop instruction with P3 specified is executed in the event task.
- When the event task is executed, P2 and P3 in task period 2 of the periodic task are executed after processing of the event task is completed.
- As a result, P3 in task period 2 of the periodic task is disabled because it comes after execution of the PrgStop instruction.
- Thereafter, P3 remains disabled even if you do not execute the PrgStop instruction with P3 specified.



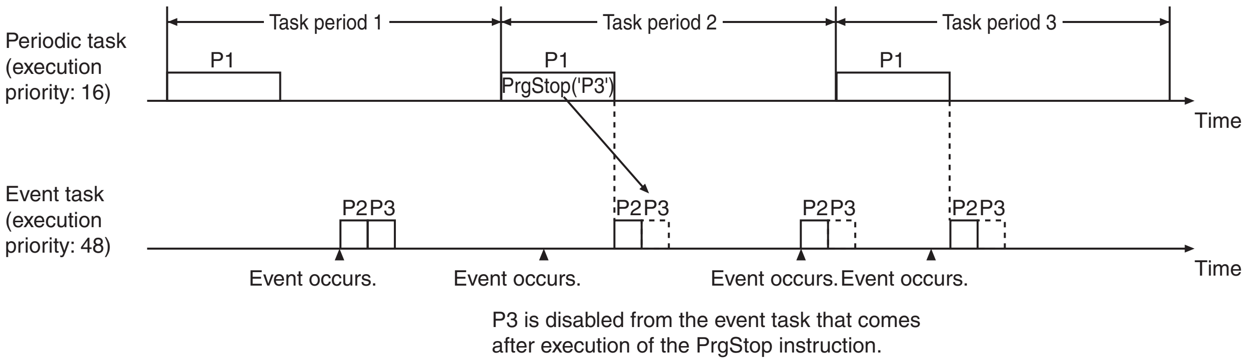
Disabling a Program in a Task with a Higher Execution Priority from an Event Task

- There are three programs in this example. P1 and P2 are in the primary periodic task, and P3 is in an event task.
- P2 is executed in task period 1 of the primary periodic task.
- The PrgStop instruction with P2 specified is executed in the event task.
- P2 is disabled from task period 2 of the primary periodic task, which is executed after the PrgStop instruction is executed.
- Thereafter, P2 remains disabled even if you do not execute the PrgStop instruction with P2 specified.



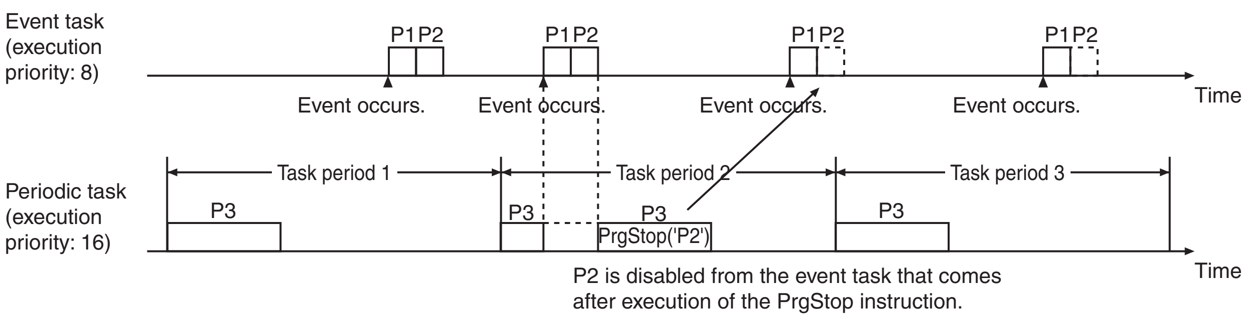
Disabling a Program in an Event Task with a Lower Execution Priority from a Periodic Task

- There are three programs in this example. P1 is in a periodic task (execution priority: 16), and P2 and P3 are in an event task (execution priority: 48).
- P3 is executed in the event task.
- The PrgStop instruction with P3 specified is executed in the periodic task.
- P3 in the event task is disabled from the event task that is executed after the PrgStop instruction is executed.
- Thereafter, P3 remains disabled even if you do not execute the PrgStop instruction with P3 specified.



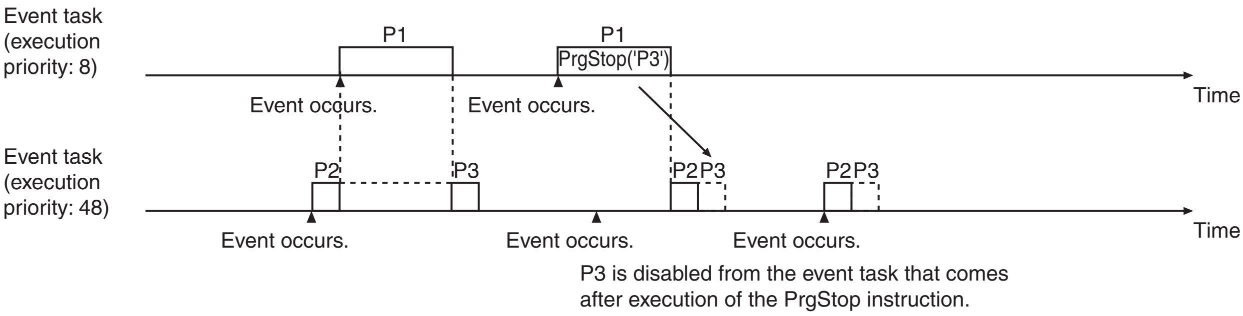
Disabling a Program in an Event Task with a Higher Execution Priority from a Periodic Task

- There are three programs in this example. P1 and P2 are in an event task (execution priority: 8), and P2 is in a periodic task (execution priority: 16).
- P2 is executed in the event task.
- The PrgStop instruction with P2 specified is executed in the periodic task.
- P2 in the event task is disabled from the event task that is executed after the PrgStop instruction is executed.
- Thereafter, P2 remains disabled even if you do not execute the PrgStop instruction with P2 specified.



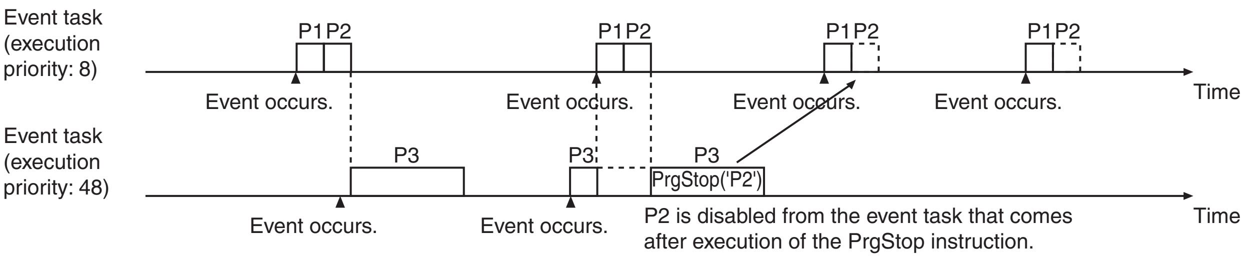
Disabling a Program in an Event Task with a Lower Execution Priority from an Event Task

- There are three programs in this example. P1 is in an event task (execution priority: 8), and P2 and P3 are in an event task (execution priority: 48).
- P3 in the event task (execution priority: 48) is executed.
- The PrgStop instruction with P3 specified is executed in the event task (execution priority: 8).
- P3 in the event task (execution priority: 48) is disabled from the event task (execution priority: 48) that is executed after the PrgStop instruction is executed.
- Thereafter, P3 remains disabled even if you do not execute the PrgStop instruction with P3 specified.



Disabling a Program in an Event Task with a Higher Execution Priority from an Event Task

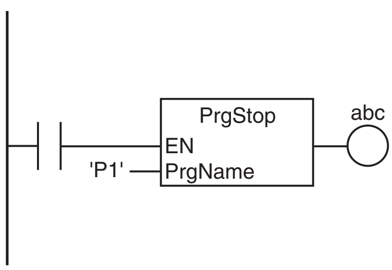
- There are three programs in this example. P1 and P2 are in an event task (execution priority: 8), and P3 is in an event task (execution priority: 48).
- P2 in the event task (execution priority: 8) is executed.
- The PrgStop instruction with P2 specified is executed in the event task (execution priority: 48).
- P2 in the event task (execution priority: 8) is disabled from the event task (execution priority: 8) that is executed after the PrgStop instruction is executed.
- Thereafter, P2 remains disabled even if you do not execute the PrgStop instruction with P2 specified.



Notation Example

The following example shows the notation for specifying disabling program P1.

LD



ST

```
abc:=PrgStop('P1');
```

Additional Information

- Use the PrgStart instruction (page 2-872) to enable a specified program from the user program.
- Use the PrgStatus instruction (page 2-901) to read the status of a specified program from the user program.

Precautions for Correct Use

- An error will not occur even if you specify a program that is already in a disabled state and execute this instruction.
- If the PrgStop instruction is executed after executing the PrgStart instruction for the same program and it is executed before the program is actually executed, the program is not executed.
- If the PrgStart instruction is executed after executing the PrgStop instruction for the same program and it is executed before the execution timing for the program, the program is not disabled.
- Processing for instructions that have an *Execute* input variable is continued until it is completed even if the execution time exceeds the task period. Before you disable programs that have such instructions, check the values of the *Busy* output variables from the instructions first to make sure that they are FALSE (i.e., to make sure that instruction execution is not in progress).
- The execution of the NX_DOutTimeStamp or NX_AryDOutTimeStamp instruction sometimes requires more than one task. Before you disable programs that have these instructions, check the values of the *Enable* input variables to the instructions first to make sure that they are FALSE.
- The operation of the programs immediately after the operating mode of the CPU Unit changes to RUN mode is controlled by the setting of the *Initial Status* for each program on the Sysmac Studio. In other words, the results of executing the PrgStart or PrgStop instruction before changing to RUN mode are not valid.
- If this instruction is executed for a program in a different task, the timing of disabling the specified program will depend on the task execution priority of both tasks. In some cases, the Controller may perform unexpected operation. You can execute this instruction in the first program in the task to which the specified program is assigned to make sure that the specified program is disabled in the same task period as the instruction.
- Confirm the following for the specified program before you execute this instruction.
  - The execution of a motion control instruction is not still in progress.
  - Processing for instructions that have an *Execute* input variable, i.e., instructions for which execution is continued until processing is completed even if the execution time exceeds the task period, is not still in progress.
  - There are no time stamp instructions that are still waiting for the specified time.
- Program outputs are not reset when the specified program is disabled. The values from before the execution is disabled are retained. If you need to reset the outputs when the program is disabled, use master control within the specified program to reset them in advance.
- Even if you disable a program with this instruction, processing for any function block instruction with an *Execute* input variable in the program is continued to the end.
- Even if you disable a program with this instruction, processing for any motion control instructions in the program is continued to the end.
- An error will occur in the following case. *Out* will be FALSE.
  - The program specified by *PrgName* does not exist.

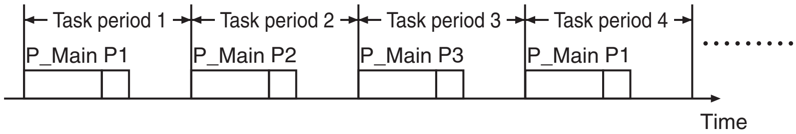
Version Information

A CPU Unit with unit version 1.08 or later and Sysmac Studio version 1.09 or higher are required to use this instruction.

Sample Programming

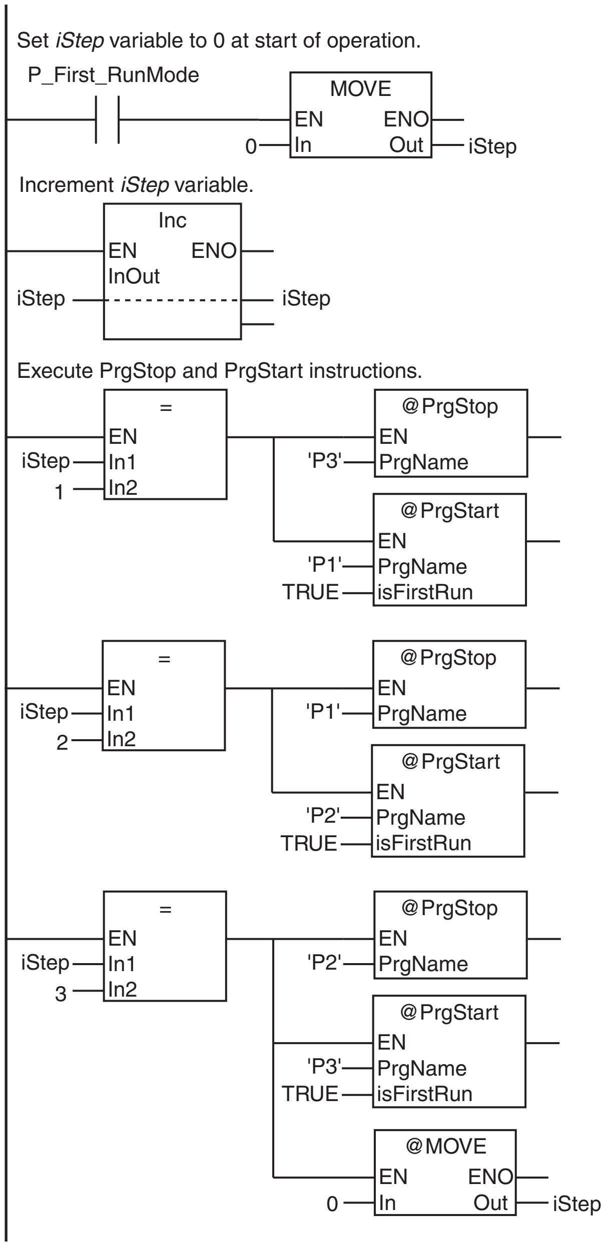
● Example in Which One of Three Programs Is Executed in Each Consecutive Task Period

In this example, there are three programs, P1, P2, and P3. One of each of these programs is executed in each consecutive task period and then they are repeated. Instructions are executed in the P_Main program to enable and disable these three programs.



LD

Variable	Data type	Default	Comment
iStep	DINT	0	Number of program to execute



ST

Variable	Data type	Default	Comment
iStep	DINT	0	Number of program to execute

```
// Set iStep variable to 0 at start of operation.
IF P_First_RunMode THEN
   iStep:=0;
END_IF;

// Increment iStep variable.
iStep:=iStep+1;

// Execute PrgStop and PrgStart instructions.
IF iStep = 1 THEN
   PrgStop('P3');
   PrgStart('P1',TRUE);
ELSIF iStep = 2 THEN
   PrgStop('P1');
   PrgStart('P2',FALSE);
ELSIF iStep = 3 THEN
   PrgStop('P2');
   PrgStart('P3',TRUE);
   iStep:=0;
END_IF;
```

● Example of Executing Only the Specified Program or Programs the Next Time Operation Starts

In this example, the program or programs to execute the next time operation starts are specified before the power supply to the Controller is turned OFF. When the power supply is next turned ON, only the specified program or programs are executed.

Programs, Modules, and Module Configuration

There are eight programs from Program 1 to Program 8.

Each program belongs to one of five modules from Module A to Module E.

Module	Programs in module
Module A	Program 1
Module B	Program 2
Module C	Program 3 and Program 4
Module D	Program 5, Program 6, and Program 7
Module E	Program 8

The programs to execute are specified by specifying a module. A combination of modules to execute is called a module configuration.

For example, if a module configuration to execute Module A and Module C was specified, Program 1, Program 3, and Program 4 would be executed.

Specifying Module Configurations to Execute

The module configurations are given with text data in a configuration file. The file name of the configuration file is Config.txt, and it is stored in the root directory of an SD Memory Card. The configuration file can contain more than one module configuration.

Before the power supply is turned OFF, a touch panel is used to specify the module configuration to execute next from the contents of the configuration file.

Format of Configuration File

The format of the configuration file is given in the following table.

Row	Contents
Row 1	Number of module configurations
Row 2 and higher	Module configuration number, Module A execution flag,*1 Module B execution flag, Module C execution flag, Module D execution flag, Module E execution flag

*1 The module is executed if the flag is TRUE and not executed if the flag is FALSE.

An example of the contents of a configuration file is given below.

```
3
Config1, TRUE, TRUE, TRUE, FALSE, FALSE
Config2, TRUE, TRUE, FALSE, TRUE, FALSE
Config3, TRUE, TRUE, TRUE, FALSE, TRUE
```

This configuration file contains three configurations, Config1, Config2, and Config3. Of these, the Config1 module configuration says to execute Module A, Module B, and Module C and to not execute Module D and Module E.

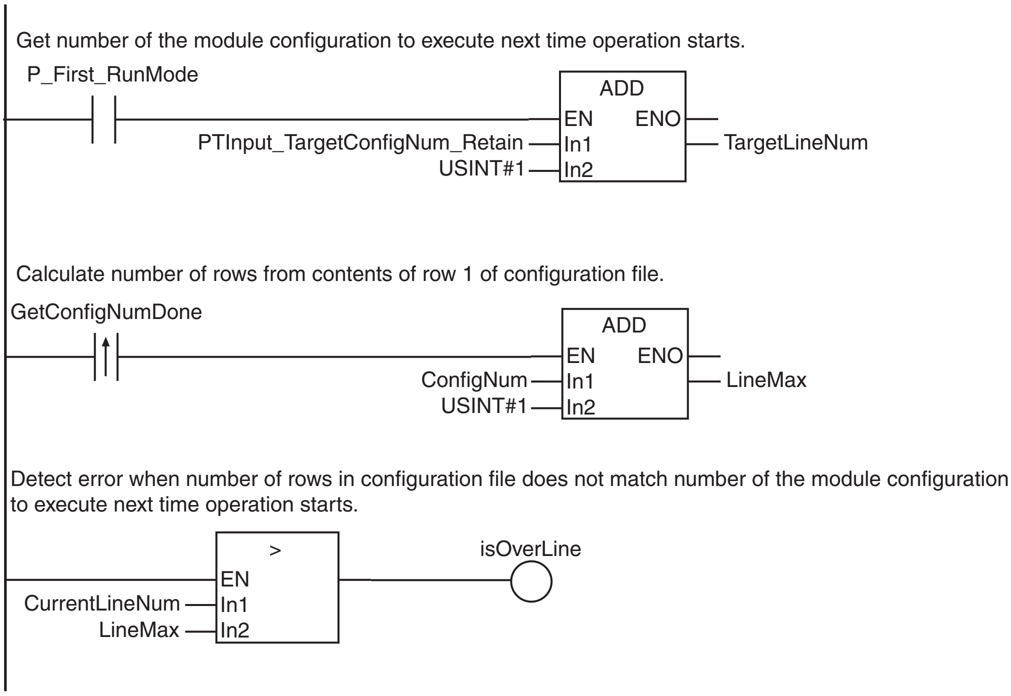
Data Type Definitions

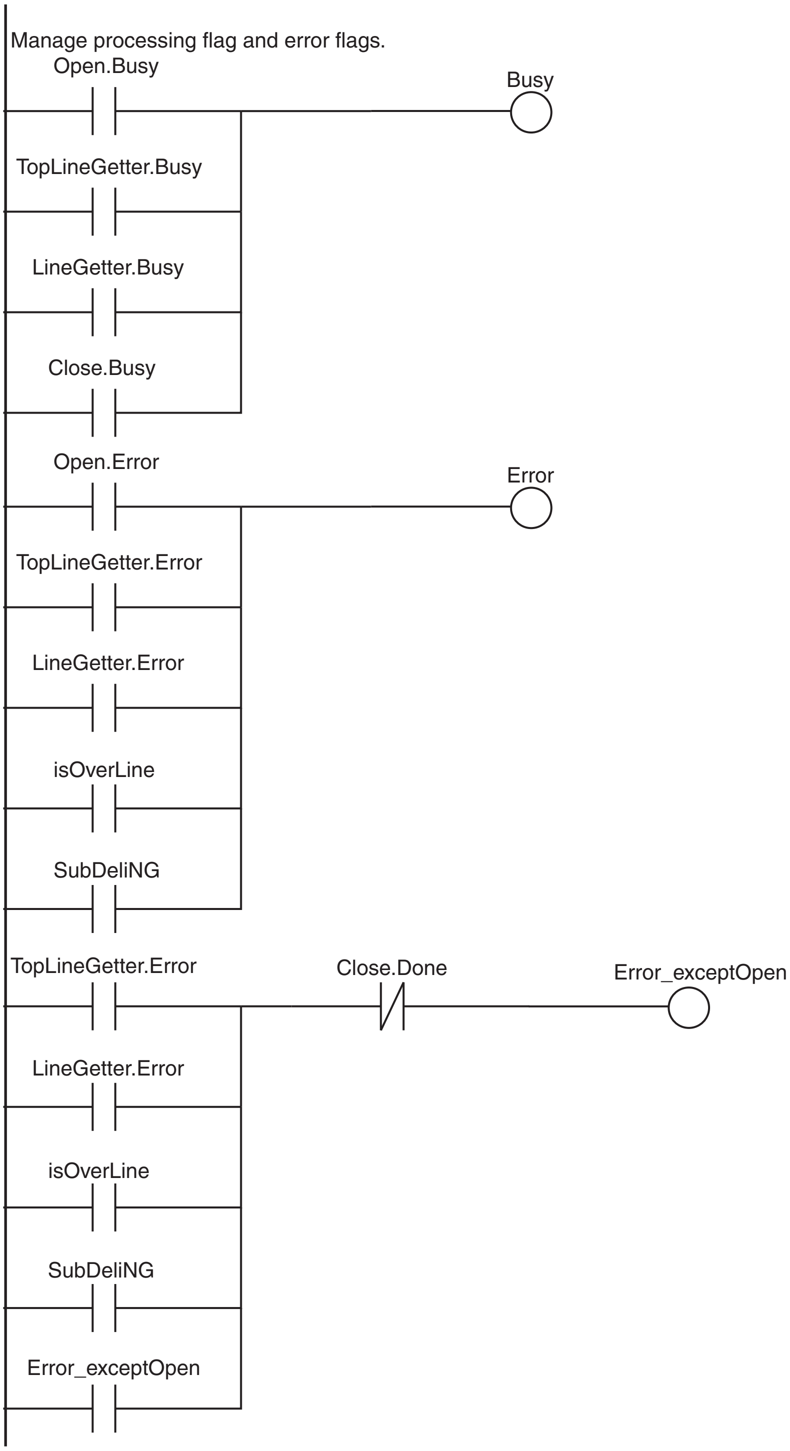
A structure called *myConfig* is defined as shown in the following table.

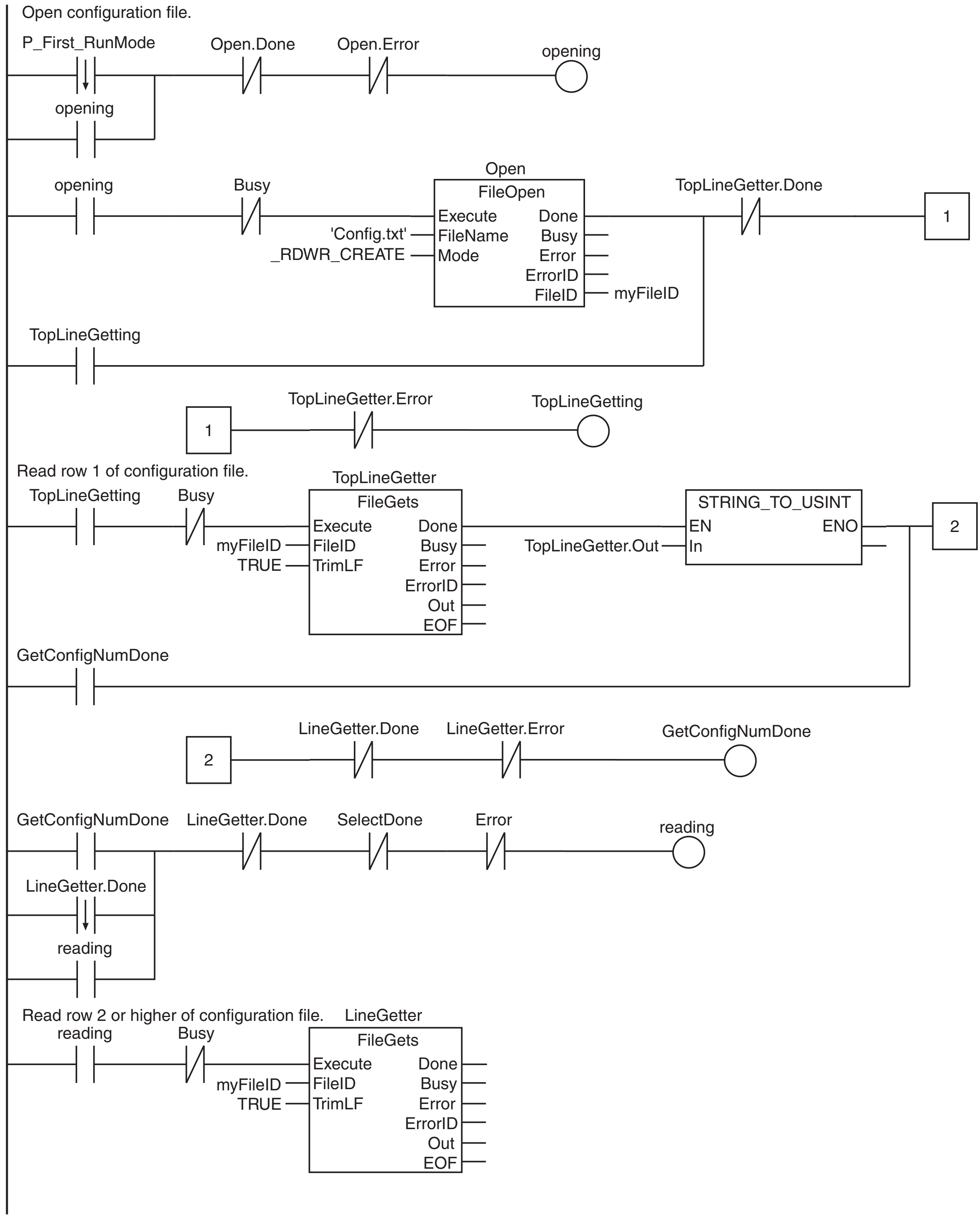
Structure		Variable	Data type	Offset type	Comment
	▼	myConfig	STRUCT	NJ	Module configuration
		configName	STRING[32]		Module configuration name
		moduleA	BOOL		Module A execution flag
		moduleB	BOOL		Module B execution flag
		moduleC	BOOL		Module C execution flag
		moduleD	BOOL		Module D execution flag
		moduleE	BOOL		Module E execution flag

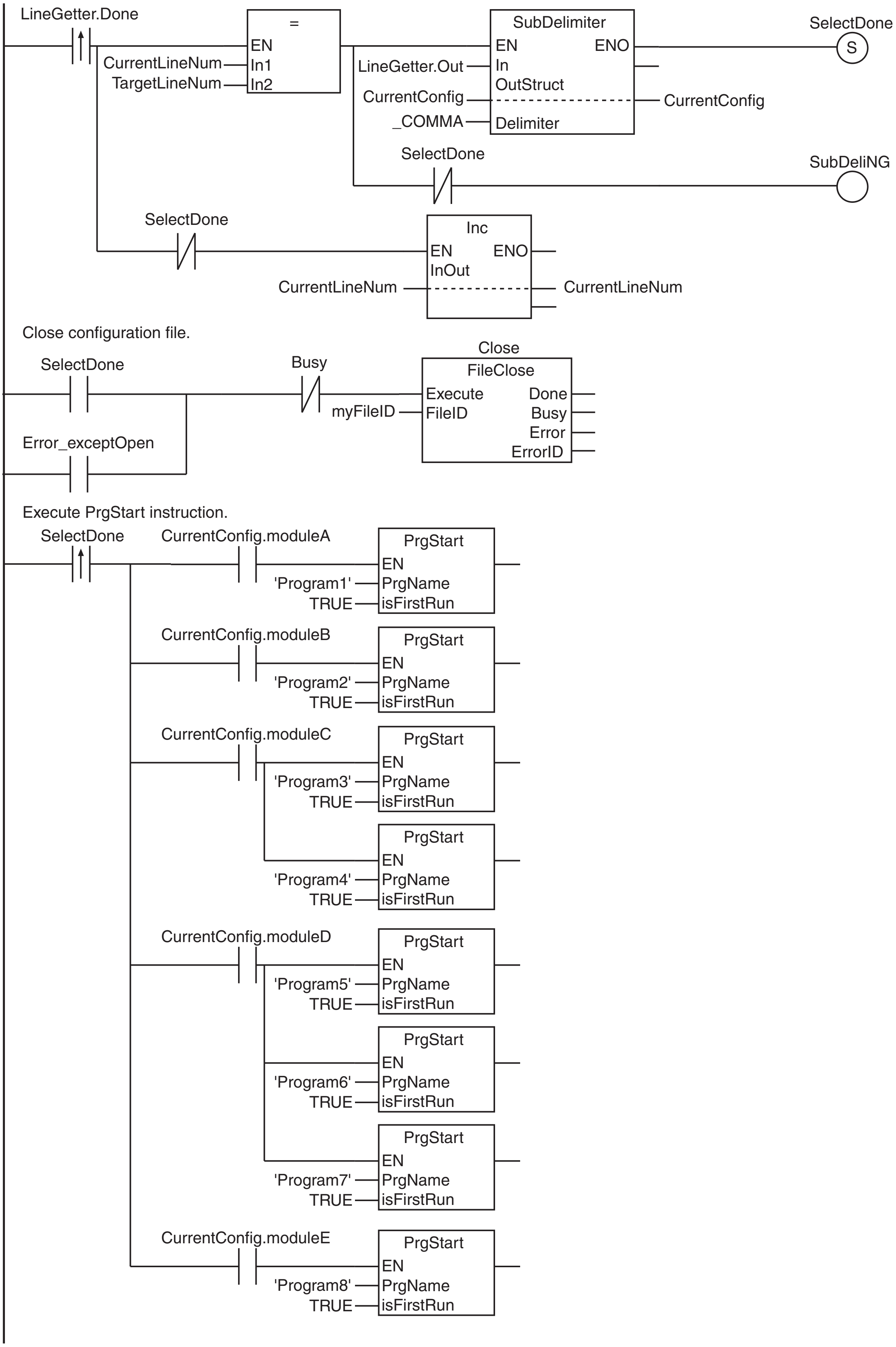
LD

Variable	Data type	Default	Retain	Comment
Open	FileOpen		---	Instance of FileOpen instruction
TopLineGetter	FileGets		---	Instance of FileGets instruction
LineGetter	FileGets		---	Instance of FileGets instruction
Close	FileClose		---	Instance of FileClose instruction
PTInput_TargetConfig-Num_Retain	USINT	0	✔	Number of the module configuration to execute next time operation starts
CurrentLineNum	USINT	1	---	Current configuration file row
TargetLineNum	USINT	0	---	Row for *CurrentConfig* in configuration file
ConfigNum	USINT	1	---	Number given in row 1 of configuration file
LineMax	USINT	3	---	Number of rows in configuration file obtained from *ConfigNum*
isOverLine	BOOL	FALSE	---	Error flag when value of *PTInput_TargetConfigNum_Retain* is larger than value of *LineMax*
Busy	BOOL	FALSE	---	Processing flag
SubDeliNG	BOOL	FALSE	---	Read error end flag for *CurrentConfig*
Error	BOOL	FALSE	---	Error flag
opening	BOOL	FALSE	---	Configuration file open execution flag
myFileID	DWORD	0	---	File ID of configuration file
TopLineGetting	BOOL	FALSE	---	*ConfigNum* read execution flag
GetConfigNumDone	BOOL	FALSE	---	*ConfigNum* read done flag
SelectDone	BOOL	FALSE	---	*CurrentConfig* read done flag
reading	BOOL	FALSE	---	Configuration file row 2 or higher read execution flag
CurrentConfig	myConfig	(configName:='', moduleA:=FALSE, moduleB:=FALSE, moduleC:=FALSE, moduleD:=FALSE)	---	Module configuration to execute next time operation starts
Error_exceptOpen	BOOL	FALSE	---	Configuration file close execution flag when error occurs


Get number of the module configuration to execute next time operation starts.
P_First_RunMode
ADD
EN
ENO
PTInput_TargetConfigNum_Retain
In1
TargetLineNum
USINT#1
In2
Calculate number of rows from contents of row 1 of configuration file.
GetConfigNumDone
ADD
EN
ENO
ConfigNum
In1
LineMax
USINT#1
In2
Detect error when number of rows in configuration file does not match number of the module configuration to execute next time operation starts.
>
isOverLine
EN
CurrentLineNum
In1
LineMax
In2


Manage processing flag and error flags.
Open.Busy
Busy
TopLineGetter.Busy
LineGetter.Busy
Close.Busy
Open.Error
Error
TopLineGetter.Error
LineGetter.Error
isOverLine
SubDeliNG
TopLineGetter.Error
Close.Done
Error_exceptOpen
LineGetter.Error
isOverLine
SubDeliNG
Error_exceptOpen


Open configuration file.
P_First_RunMode
Open.Done
Open.Error
opening
opening
opening
Busy
Open
FileOpen
Execute
Done
'Config.txt'
FileName
Busy
_RDWR_CREATE
Mode
Error
ErrorID
FileID
myFileID
TopLineGetter.Done
1
TopLineGetting
1
TopLineGetter.Error
TopLineGetting
Read row 1 of configuration file.
TopLineGetting
Busy
TopLineGetter
FileGets
Execute
Done
myFileID
FileID
Busy
TRUE
TrimLF
Error
ErrorID
Out
EOF
STRING_TO_USINT
EN
ENO
TopLineGetter.Out
In
2
GetConfigNumDone
2
LineGetter.Done
LineGetter.Error
GetConfigNumDone
GetConfigNumDone
LineGetter.Done
SelectDone
Error
reading
LineGetter.Done
reading
Read row 2 or higher of configuration file.
LineGetter
reading
Busy
FileGets
Execute
Done
myFileID
FileID
Busy
TRUE
TrimLF
Error
ErrorID
Out
EOF


LineGetter.Done
=
EN
CurrentLineNum
In1
TargetLineNum
In2
SubDelimiter
EN
ENO
LineGetter.Out
In
OutStruct
CurrentConfig
CurrentConfig
_COMMA
Delimiter
SelectDone
S
SelectDone
SubDeliNG
SelectDone
Inc
EN
ENO
InOut
CurrentLineNum
CurrentLineNum
Close configuration file.
SelectDone
Busy
Close
FileClose
Execute
Done
myFileID
FileID
Busy
Error
ErrorID
Error_exceptOpen
Execute PrgStart instruction.
SelectDone
CurrentConfig.moduleA
PrgStart
EN
'Program1'
PrgName
TRUE
isFirstRun
CurrentConfig.moduleB
PrgStart
EN
'Program2'
PrgName
TRUE
isFirstRun
CurrentConfig.moduleC
PrgStart
EN
'Program3'
PrgName
TRUE
isFirstRun
PrgStart
EN
'Program4'
PrgName
TRUE
isFirstRun
CurrentConfig.moduleD
PrgStart
EN
'Program5'
PrgName
TRUE
isFirstRun
PrgStart
EN
'Program6'
PrgName
TRUE
isFirstRun
PrgStart
EN
'Program7'
PrgName
TRUE
isFirstRun
CurrentConfig.moduleE
PrgStart
EN
'Program8'
PrgName
TRUE
isFirstRun

ST

Variable	Data type	Default	Retain	Comment
Open	FileOpen		---	Instance of FileOpen instruction
TopLineGetter	FileGets		---	Instance of FileGets instruction
LineGetter	FileGets		---	Instance of FileGets instruction
Close	FileClose		---	Instance of FileClose instruction
PTInput_TargetConfig-Num_Retain	USINT	0	✔	Number of the module configuration to execute next time operation starts
CurrentLineNum	USINT	1	---	Current configuration file row
TargetLineNum	USINT	0	---	Row for *CurrentConfig* in configura-tion file
ConfigNum	USINT	1	---	Number given in row 1 of configura-tion file
LineMax	USINT	3	---	Number of rows in configuration file obtained from *ConfigNum*
isOverLine	BOOL	FALSE	---	Error flag when value of *PTInput_Tar-getConfigNum_Retain* is larger than value of *LineMax*
Busy	BOOL	FALSE	---	Processing flag
SubDeliNG	BOOL	FALSE	---	Read error end flag for *CurrentConfig*
Error	BOOL	FALSE	---	Error flag
opening	BOOL	FALSE	---	Configuration file open execution flag
myFileID	DWORD	0	---	File ID of configuration file
TopLineGetting	BOOL	FALSE	---	*ConfigNum* read execution flag
GetConfigNumDone	BOOL	FALSE	---	*ConfigNum* read done flag
SelectDone	BOOL	FALSE	---	*CurrentConfig* read done flag
reading	BOOL	FALSE	---	Configuration file row 2 or higher read execution flag
CurrentConfig	myConfig	(configName:='', moduleA:=FALSE, moduleB:=FALSE, moduleC:=FALSE, moduleD:=FALSE)	---	Module configuration to execute next time operation starts
Error_exceptOpen	BOOL	FALSE	---	Configuration file close execution flag when error occurs
R_GetConfigNumDone	R_TRIG		---	Instance of R_TRIG instruction
RS_1	RS		---	Instance of RS instruction
RS_2	RS		---	Instance of RS instruction
SecondCycle	F_TRIG		---	Instance of F_TRIG instruction
RS_3	RS		---	Instance of RS instruction
ConvertDone	BOOL	FALSE	---	Conversion done flag for converting character in row 1 of configuration file to a number.
RS_4	RS		---	Instance of RS instruction
F_LineGetterDone	F_TRIG		---	Instance of F_TRIG instruction
R_LineGetterDone	R_TRIG		---	Instance of R_TRIG instruction
isTargetLine	BOOL	FALSE	---	Flag to indicate that current row is the row of the module configuration to execute next time operation starts
SubDeliCondition	BOOL	FALSE	---	Expansion execution flag from mod-ule configuration to *CurrentConfig*.

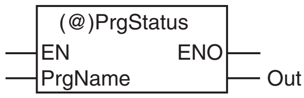
Variable	Data type	Default	Retain	Comment
RS_5	RS		---	Instance of RS instruction
SubDeliDone	BOOL	FALSE	---	Expansion done flag from module configuration to *CurrentConfig*.
R_SelectDone	R_TRIG		---	Instance of R_TRIG instruction

```
// Get number of the module configuration to execute next time operation starts.
IF P_First_RunMode THEN
   TargetLineNum := PTInput_TargetConfigNum_Retain + USINT#1;
END_IF;

// Calculate number of rows from contents of row 1 of configuration file.
R_GetConfigNumDone(Clk:=GetConfigNumDone);
IF R_GetConfigNumDone.Q THEN
   LineMax := ConfigNum + USINT#1;
END_IF;

// Detect error when number of rows in configuration file does not match number
// of the module configuration to execute next time operation starts.
isOverLine := (CurrentLineNum > LineMax);

// Manage processing flag and error flags.
Busy := Open.Busy OR TopLineGetter.Busy OR LineGetter.Busy OR Close.Busy;

Error := Open.Error OR TopLineGetter.Error OR LineGetter.Error OR isOverLine
OR SubDeliNG;

RS_1(Set:= (TopLineGetter.Error OR LineGetter.Error OR isOverLine OR SubDeliNG),
reset1 := Close.Done, Q1 => Error_exceptOpen);

// Open configuration file.
SecondCycle(Clk:=P_First_RunMode);
RS_2(Set := SecondCycle.Q, reset1:=(Open.Done OR Open.Error), Q1 => opening);
Open(Execute:=(opening & NOT(Busy)), FileName :='Config.txt', FileID => myFil-
eID);
RS_3(Set  :=  Open.Done,  Reset1:=(TopLineGetter.Done  OR  TopLineGetter.Error),
Q1=>TopLineGetting);

// Read row 1 of configuration file.
TopLineGetter(Execute :=(TopLineGetting & NOT(Busy)), FileID := myFileID, TrimLF
:= TRUE);
ConfigNum  :=  STRING_TO_USINT(EN:=  TopLineGetter.Done,  IN:=TopLineGetter.Out,
ENO=>ConvertDone);
RS_4(Set := ConvertDone, Reset1:=(LineGetter.Done OR LineGetter.Error), Q1=>Get-
ConfigNumDone);
F_LineGetterDone(Clk:=LineGetter.Done);
RS_5(Set := (GetConfigNumDone OR F_LineGetterDone.Q), Reset1:=(LineGetter.Done OR
SelectDone OR Error), Q1=>reading);

// Read row 2 or higher of configuration file.
LineGetter(Execute:=(reading & NOT(Busy)), FileID:=myFileID, TrimLF := TRUE);
R_LineGetterDone(Clk:=LineGetter.Done);
isTargetLine := (CurrentLineNum = TargetLineNum);
SubDeliCondition := (R_LineGetterDone.Q & isTargetLine);
SubDelimiter(EN := SubDeliCondition, In := LineGetter.Out, OutStruct := Current-
Config, Delimiter := _COMMA, ENO => SubDeliDone);
IF SubDeliDone THEN
   SelectDone := TRUE;
END_IF;
SubDeliNG := (SubDeliCondition & NOT(SubDeliDone));
Inc(EN := (R_LineGetterDone.Q & NOT(SelectDone)), InOut:= CurrentLineNum);
```

```
// Close configuration file.
Close(Execute := ((SelectDone OR Error_exceptOpen) & NOT(Busy)), FileID := myFil-
eID);

// Execute PrgStart instruction.
R_SelectDone(Clk:=SelectDone);
//moduleA
PrgStart(EN := (R_SelectDone.Q & CurrentConfig.moduleA), PrgName :='Program1',
isFirstRun:=TRUE);
//moduleB
PrgStart(EN := (R_SelectDone.Q & CurrentConfig.moduleB), PrgName :='Program2',
isFirstRun:=TRUE);
//moduleC
PrgStart(EN := (R_SelectDone.Q & CurrentConfig.moduleC), PrgName :='Program3',
isFirstRun:=TRUE);
PrgStart(EN := (R_SelectDone.Q & CurrentConfig.moduleC), PrgName :='Program4',
isFirstRun:=TRUE);
//moduleD
PrgStart(EN := (R_SelectDone.Q & CurrentConfig.moduleD), PrgName :='Program5',
isFirstRun:=TRUE);
PrgStart(EN := (R_SelectDone.Q & CurrentConfig.moduleD), PrgName :='Program6',
isFirstRun:=TRUE);
PrgStart(EN := (R_SelectDone.Q & CurrentConfig.moduleD), PrgName :='Program7',
isFirstRun:=TRUE);
//moduleE
PrgStart(EN := (R_SelectDone.Q & CurrentConfig.moduleE), PrgName :='Program8',
isFirstRun:=TRUE);
```

PrgStatus

The PrgStatus instruction reads the status of the specified program.

Instruction	Name	FB/FUN	Graphic expression	ST expression
PrgStatus	Read Program Status	FUN		Out:=PrgStatus(PrgName);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
PrgName	Program name	Input	Name of specified program	128 bytes max. (127 single-byte alphanumeric characters plus the final NULL character)	---	*1
Out	Program status	Output	Status of program the next time the timing for execution occurs TRUE: Enabled. FALSE: Disabled.	Depends on data type.	---	---

*1 If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
PrgName																				OK
Out	OK																			

Function

The PrgStatus instruction reads the status of the program specified with *PrgName* for the next time the timing for executing the program occurs. The value of *Out* is TRUE if the specified program will be enabled the next time the timing for executing it occurs. The value of *Out* is FALSE if the specified program will be disabled the next time the timing for executing it occurs.

The following table shows the meaning of “enabled” and “disabled” for the next time the timing for executing a program occurs.

Program status	Description
Enabled the next time the timing for execution occurs	• The *Initial Status* for the relevant program is set to *Run* on the Sysmac Studio. • The PrgStart instruction was executed for the program.
Disabled the next time the timing for execution occurs	• The *Initial Status* for the relevant program is set to *Stop* on the Sysmac Studio. • The PrgStop instruction was executed for the program.

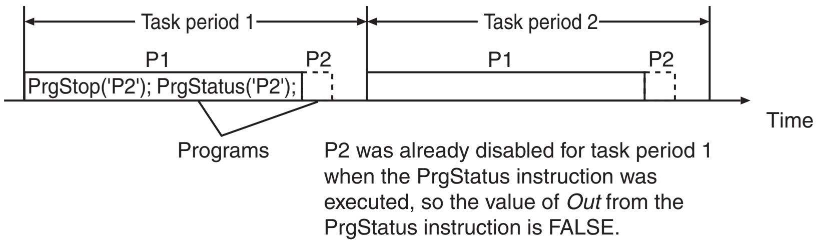
The specified program can be in the same task as this instruction, or it can be in a different task.

Operation Example

This section provides some examples of the operation of this instruction.

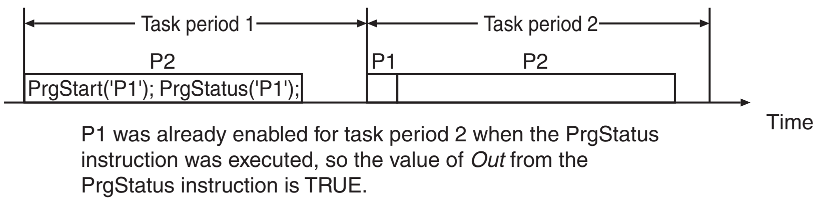
● Reading the Status of a Program After the PrgStatus Instruction in the Current Task

- In this example, there are two programs, P1 and P2, in the same task.
- The PrgStop instruction with P2 specified is executed in P1 of task period 1.
- The PrgStatus instruction with P2 specified is then executed in P1 of task period 1.
- P2 was disabled for task period 1, so the value of *Out* from the PrgStatus instruction is FALSE.



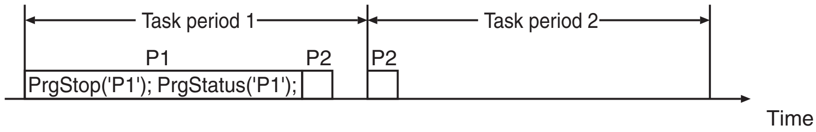
● Reading the Status of a Program Before the PrgStatus Instruction in the Current Task

- In this example, there are two programs, P1 and P2, in the same task.
- The PrgStart instruction with P1 specified is executed in P2 of task period 1.
- The PrgStatus instruction with P1 specified is then executed in P2 of task period 1.
- P1 was enabled for task period 2, so the value of *Out* from the PrgStatus instruction is TRUE.



Reading the Status of the Program That Includes the PrgStatus Instruction

- The PrgStop instruction with P1 specified is executed in P1 of task period 1.
- The PrgStatus instruction with P1 specified is then executed in P1 of task period 1.
- P1 was disabled for task period 2, so the value of *Out* from the PrgStatus instruction is FALSE.

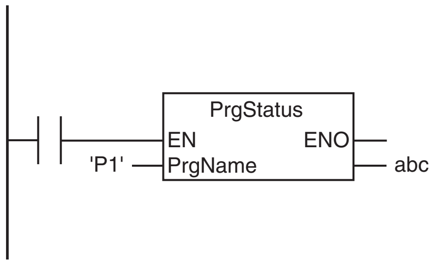


P1 was already disabled for task period 2 when the PrgStatus instruction was executed, so the value of *Out* from the PrgStatus instruction is FALSE.

Notation Example

The following example shows the notation for reading the status of the P1 program.

LD



ST

```
abc:=PrgStatus('P1');
```

Additional Information

- Use the PrgStart instruction (page 2-872) to enable a specified program from the user program.
- Use the PrgStop instruction (page 2-881) to disable a specified program from the user program.

Precautions for Correct Use

- An error will occur in the following case. *Out* will be FALSE.
  - The program specified by *PrgName* does not exist.

Version Information

A CPU Unit with unit version 1.08 or later and Sysmac Studio version 1.09 or higher are required to use this instruction.

Sample Programming

In this example, there are three programs, P1, P2, and P3. Operations on a touch panel are used to change the program to execute.

Touch Panel Specifications

This example assumes that a touch panel is connected to the Controller.

The touch panel has the following lamps.

Lamp name	Description
P1 executing lamp	Lit when P1 execution is in progress.
P2 executing lamp	Lit when P2 execution is in progress.
P3 executing lamp	Lit when P3 execution is in progress.

The touch panel also has the following buttons.

Button name	Operation when button is pressed
Execution program change button	Each time this button is pressed, the program to execute changes in order from P1 to P2 to P3, and then returns to P1.

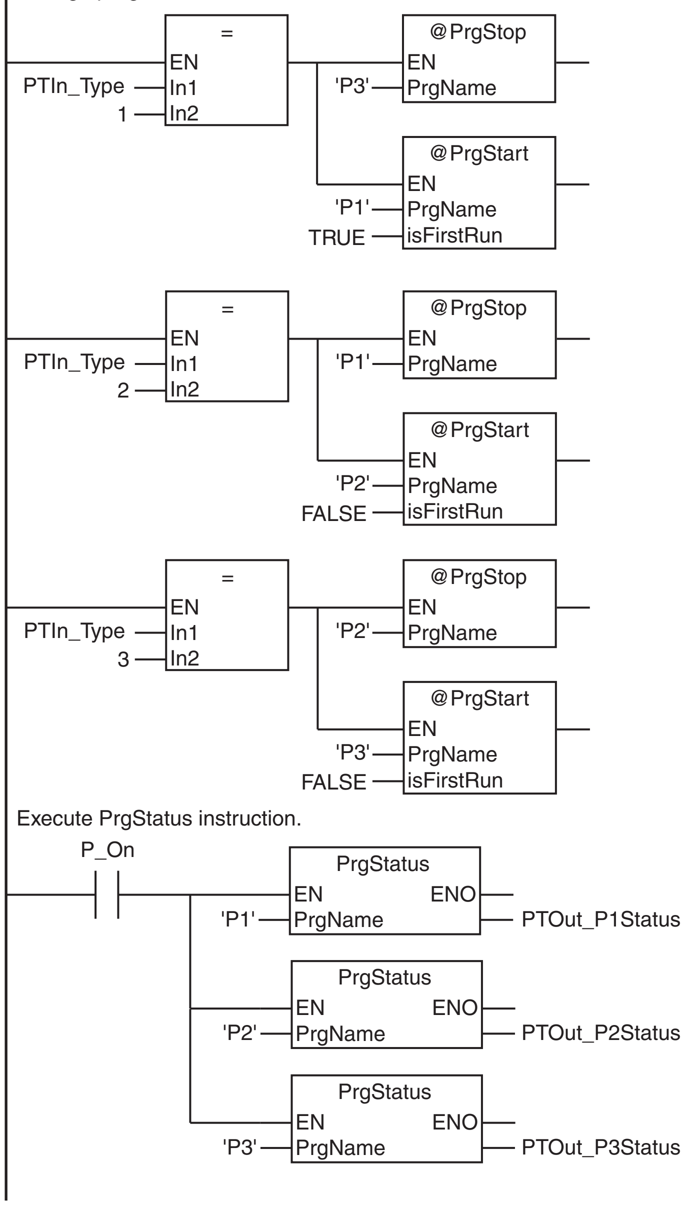
Global Variables

Variable	Data type	Initial value	Comment
PTIn_Type	INT	0	Execution program change button input
PTOut_P1Status	BOOL	FALSE	P1 executing lamp output
PTOut_P2Status	BOOL	FALSE	P2 executing lamp output
PTOut_P3Status	BOOL	FALSE	P3 executing lamp output

LD

External Variables	Variable	Data type	Comment
	PTIn_Type	INT	Execution program change button input
	PTOut_P1Status	BOOL	P1 executing lamp output
	PTOut_P2Status	BOOL	P2 executing lamp output
	PTOut_P3Status	BOOL	P3 executing lamp output

Change program to execute.



Execute PrgStatus instruction.

ST

External Variables	Variable	Data type	Comment
	PTIn_Type	INT	Execution program change button input
	PTOut_P1Status	BOOL	P1 executing lamp output
	PTOut_P2Status	BOOL	P2 executing lamp output
	PTOut_P3Status	BOOL	P3 executing lamp output

```
// Change program to execute.
IF PTIn_Type = 1 THEN
   PrgStop('P3');
   PrgStart('P1',TRUE);
ELSIF PTIn_Type = 2 THEN
   PrgStop('P1');
   PrgStart('P2',FALSE);
ELSIF PTIn_Type = 3 THEN
   PrgStop('P2');
   PrgStart('P3',FALSE);
END_IF;

// Execute PrgStatus instruction.
IF P_On THEN
   PTOut_P1Status:=PrgStatus('P1');
   PTOut_P2Status:=PrgStatus('P2');
   PTOut_P3Status:=PrgStatus('P3');
END_IF;
```

EtherCAT Communications Instructions

EC_CoESDOWrite

The EC_CoESDOWrite instruction writes a value to a CoE* object of a specified slave on an EtherCAT network.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
EC_CoES-DOWrite	Write EtherCAT CoE SDO	FB	EC_CoESDOWrite_instance EC_CoESDOWrite Execute — Done NodeAdr — Busy SdoObj — Error TimeOut — ErrorID WriteDat — AbortCode WriteSize	EC_CoESDOWrite_instance(Execute, NodeAdr, SdoObj, TimeOut, WriteDat, WriteSize, Done, Busy, Error, ErrorID, AbortCode);

* CoE stands for CAN Application Protocol over EtherCAT.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
NodeAdr	Slave node address	Input	Node address of the slave to access	1 to 512*1	---	---
SdoObj	SDO parameter		SDO parameter	---		
TimeOut	Timeout time		0: 2.0s 1 to 65535: 0.1 to 6553.5 s	Depends on data type.	0.1 s	20 (2.0 s)
WriteDat	Write data		Write data		---	---
WriteSize	Write data size		Write data size*2	1 to 2048	Bytes	
AbortCode	Abort code	Output	Response code for SDO access specified by CoE 0: Normal end	Depends on data type.	---	---

*1 The range is 1 to 192 for an NJ-series CPU Unit.

*2 The write data size may be less than 1 byte, e.g., if the write data is BOOL or a BOOL array. If it is less than 1 byte, set the value of *WriteSize* to 1.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
NodeAdr							OK													
SdoObj	Refer to *Function* for details on the structure _sSDO_ACCESS.																			
TimeOut							OK													
WriteDat	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	An enumeration, array, array element, structure member, or union member can also be specified.																			
WriteSize							OK													
AbortCode				OK																

Function

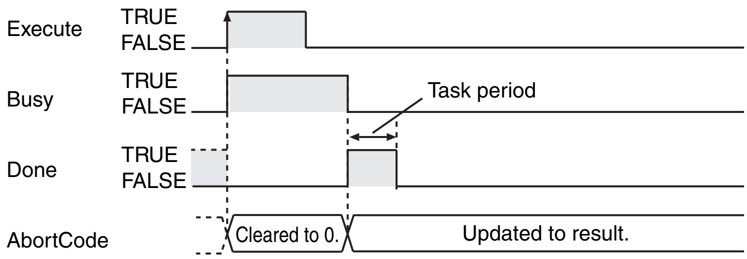
The EC_CoESDOWrite instruction writes data to the CoE object of the node specified with slave node address *NodeAdr*. The content of *WriteDat* is written to the object. The number of bytes of data to write is specified with *WriteSize*. The SDO parameter is specified with *SdoObj*.

The data type of *SdoObj* is structure _sSDO_ACCESS. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
SdoObj	SDO parameter	SDO parameter	_sSDO_ACCESS	---	---	---
Index	Index	Index number in the object dictionary defined in CoE	UINT	1 to 65535		
Subindex	Subindex	Subindex number in the object dictionary defined in CoE	USINT			
IsCompleteAccess	Complete access	Specification of complete access of SDO TRUE: Access data for all subindexes FALSE: Access data for the specified subindex	BOOL	Depends on data type.	---	---

After the write is completed, the instruction waits for the response for the time specified with timeout time *TimeOut*. The response is stored in *AbortCode*. *AbortCode* is 0 for a normal response. A value is stored in *AbortCode* only when the value of *ErrorID* is 16#1804 (SDO abort response).
The meaning and values of *AbortCode* depend on the slave. Refer to the manual for the slave.

The following figure shows a timing chart. A value is stored in *AbortCode* when *Busy* changes to FALSE after the completion of instruction processing.



Related System-defined Variables

Name	Meaning	Data type	Description
_EC_MBXSlavTbl[i] "i" is the node address.	Message Communications Enabled Slave Table	BOOL	This variable indicates whether communications are possible for each slave. TRUE: Communications are possible. FALSE: Communications are not possible.

Additional Information

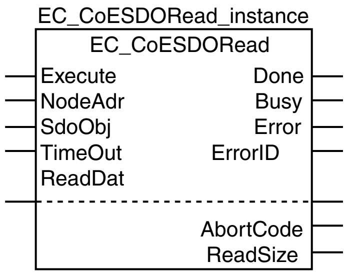
- Refer to the *NJ/NX-series CPU Unit Built-in EtherCAT Port User's Manual* (Cat. No. W505) or *NY-series Industrial Panel PC / Industrial Box PC Built-in EtherCAT Port User's Manual* (Cat. No. W562) for details on EtherCAT communications.
- Refer to *A-4 SDO Abort Codes* on page A-21 for the SDO abort codes.

Precautions for Correct Use

- Always use a variable for the input parameter to pass to *WriteDat*. A building error will occur if a constant is passed.
- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the NJ/NX-series and NY-series Controller EtherCAT ports.
- You can execute a maximum of 32 of the following instructions at the same time: EC_CoESDOWrite, EC_CoESDORead, EC_StartMon, EC_StopMon, EC_SaveMon, EC_CopyMon, EC_DisconnectSlave, EC_ConnectSlave, EC_ChangeEnableSetting, IOL_ReadObj, and IOL_WriteObj.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The EtherCAT master is not in a state that allows message communications.
  - The slave specified with *NodeAdr* does not exist.
  - The slave specified with *NodeAdr* is not in a state that allows communications.
  - The slave returns an error response.
  - More than 32 of the following instructions were executed at the same time: EC_CoESDOWrite, EC_CoESDORead, EC_StartMon, EC_StopMon, EC_SaveMon, EC_CopyMon, EC_DisconnectSlave, EC_ConnectSlave, EC_ChangeEnableSetting, IOL_ReadObj, and IOL_WriteObj.

EC_CoESDORead

The EC_CoESDORead instruction reads a value from a CoE* object of a specified slave on an EtherCAT network.

Instruction	Name	FB/FUN	Graphic expression	ST expression
EC_CoESDORead	Read EtherCAT CoE SDO	FB		EC_CoESDORead_instance(Execute, NodeAdr, SdoObj, TimeOut, ReadDat, Done, Busy, Error, ErrorID, AbortCode, ReadSize);

* CoE stands for CAN Application Protocol over EtherCAT.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
NodeAdr	Slave node address	Input	Node address of the slave to access	1 to 512*1	---	---
SdoObj	SDO parameter		SDO parameter	---		
TimeOut	Timeout time		0: 2.0s 1 to 65535: 0.1 to 6553.5 s	Depends on data type.	0.1 s	0 (2.0 s)
AbortCode	Abort code	Output	Response code for SDO access specified by CoE 0: Normal end	Depends on data type.	---	---
ReadSize	Read data size		Size of data stored in *ReadDat* after the data is read*2		Bytes	
ReadDat	Read data	In-out	Read data buffer	Depends on data type.	---	---

*1 The range is 1 to 192 for an NJ-series CPU Unit.

*2 The read data size may be less than 1 byte, e.g., if the read data is BOOL or a BOOL array. If it is less than 1 byte, set the value of *ReadSize* to 1.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
NodeAdr							OK													
SdoObj	Refer to *Function* for details on the structure _sSDO_ACCESS.																			
TimeOut							OK													
AbortCode				OK																
ReadSize							OK													
ReadDat	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	An enumeration, array, array element, structure member, or union member can also be specified.																			

Function

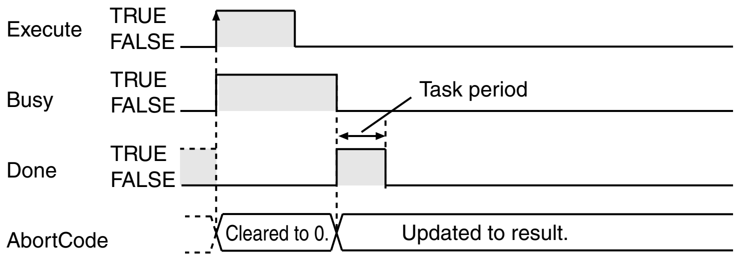
The EC_CoESDORead instruction reads data from the CoE object of the node specified with slave node address *NodeAdr*. The read data is stored in *ReadDat*. Then size of data that was stored is stored in *ReadSize*. The value of *ReadSize* is valid only when the data was stored successfully.
The SDO parameter is specified with *SdoObj*.

The data type of *SdoObj* is structure _sSDO_ACCESS. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
SdoObj	SDO parameter	SDO parameter	_sSDO_ ACCESS	---	---	---
Index	Index	Index number in the object dictionary defined in CoE	UINT	1 to 65535	---	---
Subindex	Subindex	Subindex number in the object dictionary defined in CoE	USINT	Depends on data type.		
IsCompleteAccess	Complete access	Specification of complete access of SDO TRUE:Access data for all subindexes FALSE:Access data for the specified subindex	BOOL			

After the read is completed, the instruction waits for the response for the time specified with timeout time *TimeOut*. The response is stored in *AbortCode*. *AbortCode* is 0 for a normal response. A value is stored in *AbortCode* only when the value of *ErrorID* is 16#1804 (SDO abort response).
The meaning and values of *AbortCode* depend on the slave. Refer to the manual for the slave.

The following figure shows a timing chart. A value is stored in *AbortCode* when *Busy* changes to FALSE after the completion of instruction processing.



Related System-defined Variables

Name	Meaning	Data type	Description
_EC_MBXSlaveTbl[i] "i" is the node address.	Message Communications Enabled Slave Table	BOOL	This variable indicates whether communications are possible for each slave. TRUE: Communications are possible. FALSE: Communications are not possible.

Additional Information

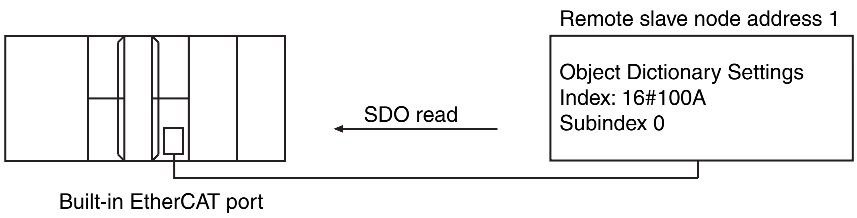
- Refer to the *NJ/NX-series CPU Unit Built-in EtherCAT Port User's Manual* (Cat. No. W505) or *NY-series Industrial Panel PC / Industrial Box PC Built-in EtherCAT Port User's Manual* (Cat. No. W562) for details on EtherCAT communications.
- Refer to *A-4 SDO Abort Codes* on page A-21 for the SDO abort codes.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the NJ/NX-series and NY-series Controller EtherCAT ports.
- You can execute a maximum of 32 of the following instructions at the same time: EC_CoESDOWrite, EC_CoESDORead, EC_StartMon, EC_StopMon, EC_SaveMon, EC_CopyMon, EC_DisconnectSlave, EC_ConnectSlave, EC_ChangeEnableSetting, IOL_ReadObj, and IOL_WriteObj.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The EtherCAT master is not in a state that allows message communications.
  - The slave specified with *NodeAdr* does not exist.
  - The slave specified with *NodeAdr* is not in a state that allows communications.
  - The slave returns an error response.
  - The read data size is larger than the size of *ReadDat*.
  - More than 32 of the following instructions were executed at the same time: EC_CoESDOWrite, EC_CoESDORead, EC_StartMon, EC_StopMon, EC_SaveMon, EC_CopyMon, EC_DisconnectSlave, EC_ConnectSlave, EC_ChangeEnableSetting, IOL_ReadObj and IOL_WriteObj.

Sample Programming

This sample uses an EtherCAT SDO message to read the software version of an OMRON 1S-series Servo Drive. The node address of the slave is 1.
The object index for the software version is 16#100A. The subindex is 0. The read value is stored in STRING variable *VersionInfo*.

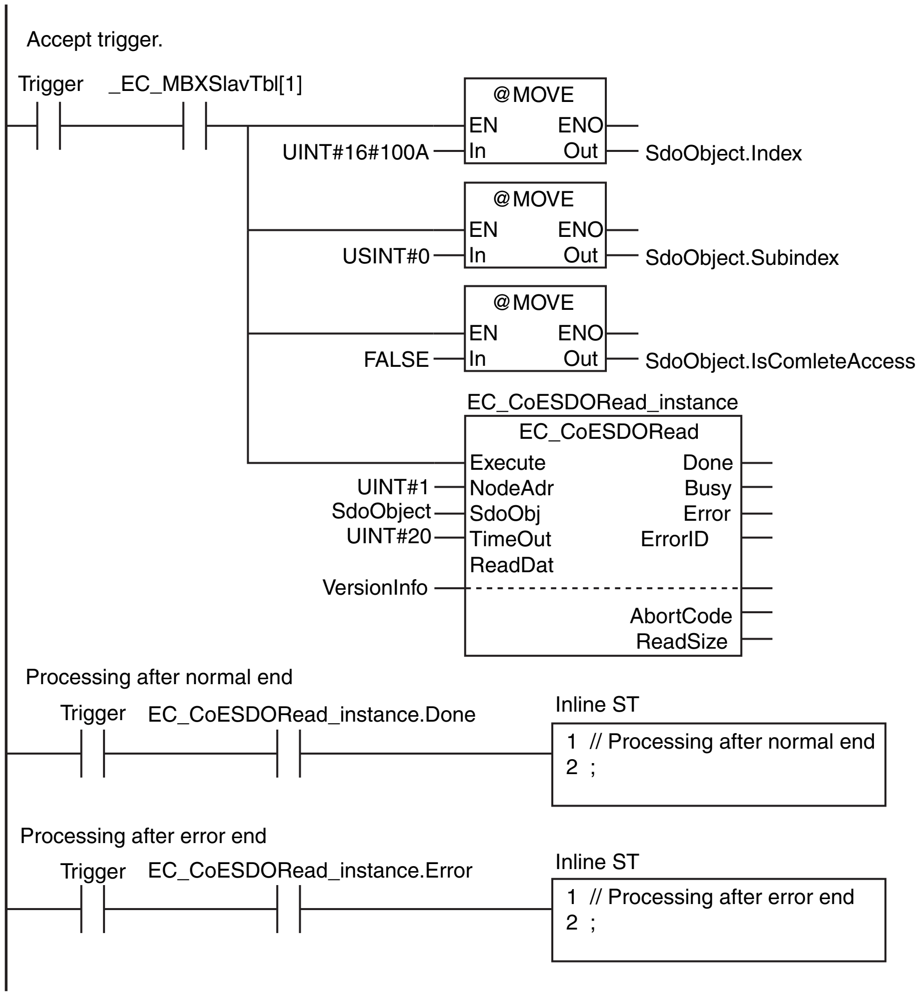


LD

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	FALSE	Execution condition
	SdoObject	_sSDO_ACCESS	(Index:=0, Subindex:=0, IsCompleteAccess:=FALSE)	SDO parameter
	VersionInfo	STRING[256]	''	Read data
	EC_CoESDORe-ad_instance	EC_CoESDORead		

External Variables	Variable	Data type	Constant	Comment
	_EC_MBXSlavTbl	ARRAY[1..512] OF BOOL*1	✔	Message Communications Enabled Slave Table

*1 The data type is ARRAY [1..192] OF BOOL for an NJ-series CPU Unit.



ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	FALSE	Execution condition
	SdoObject	_sSDO_ACCESS	(Index:=0, Subindex:=0, IsCompleteAccess:=FALSE)	SDO parameter
	DoSdoRead	BOOL	FALSE	Processing
	VersionInfo	STRING[256]	''	Read data
	NormalEnd	UINT	0	Normal end
	ErrorEnd	UINT	0	Error end
	EC_CoESDORead_instance	EC_CoESDORead		

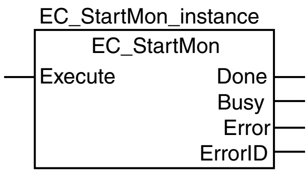
External Variables	Variable	Data type	Constant	Comment
	_EC_MBXSlavTbl	ARRAY[1..512] OF BOOL *1	✔	Message Communications Enabled Slave Table

*1 The data type is ARRAY [1..192] OF BOOL for an NJ-series CPU Unit.

```
// Detect when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (DoSdoRead=FALSE) AND (_EC_MBXSlavTbl[1]=TRUE) ) THEN
    DoSdoRead                  :=TRUE;
    SdoObject.Index            :=UINT#16#100A;
    SdoObject.Subindex         :=USINT#0;
    SdoObject.IsCompleteAccess:=FALSE;
    EC_CoESDORead_instance(
        Execute:=FALSE,           // Initialize instance.
        ReadDat:=VersionInfo);    // Dummy
END_IF;

// Execute EC_CoESDORead instruction.
IF (DoSdoRead=TRUE) THEN
    EC_CoESDORead_instance(
        Execute :=TRUE,
        NodeAdr :=UINT#1,         // Node address 1
        SdoObj  :=SdoObject,      // SDO parameter
        TimeOut :=UINT#20,        // Timeout time: 2.0 s
        ReadDat :=VersionInfo);   // Read data

    IF (EC_CoESDORead_instance.Done=TRUE) THEN
        // Processing after normal end
        NormalEnd:=NormalEnd+UINT#1;
    ELSIF (EC_CoESDORead_instance.Error=TRUE) THEN
        // Processing after error end
        ErrorEnd :=ErrorEnd+UINT#1;
    END_IF;

END_IF;
```

EC_StartMon

The EC_StartMon instruction starts execution of packet monitoring for EtherCAT communications.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
EC_StartMon	Start EtherCAT Packet Monitor	FB		EC_SatrtMon_instance(Execute, Done, Busy, Error, ErrorID);

Variables

Only common variables are used.

Function

The EC_StartMon instruction starts execution of packet monitoring for EtherCAT communications. The packet monitor function collects a specified number of the most recent EtherCAT communications packets. When the specified number of packets is exceeded, old packets are discarded in order. After the EC_StartMon instruction is executed, packet monitoring continues until the EC_StopMon instruction is executed.

Related System-defined Variables

Name	Meaning	Data type	Description
_EC_PktMonStop	Packet Monitoring Stopped	BOOL	This variable shows if packet monitoring is stopped. TRUE: Stopped. FALSE: Not stopped.
_EC_PktSaving	Saving Packet Data File	BOOL	This variable shows if the instruction is saving packet data in an internal file in the main memory of the CPU Unit. TRUE: Saving. FALSE: Not saving.

Additional Information

- You cannot save collected packet data in an internal file of the main memory of the CPU Unit during ECATStartMonitor execution.
- Do the following to save packet data in an internal file in the main memory of the CPU Unit: First, execute the EC_StopMon instruction to stop packet monitoring. Then execute the EC_SaveMon instruction to save the packets.
- Refer to the *NJ/NX-series CPU Unit Built-in EtherCAT Port User's Manual* (Cat. No. W505) or *NY-series Industrial Panel PC / Industrial Box PC Built-in EtherCAT Port User's Manual* (Cat. No. W562) for details on EtherCAT communications.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the NJ/NX-series and NY-series Controller EtherCAT ports.
- You can execute a maximum of 32 of the following instructions at the same time: EC_CoESDOWrite, EC_CoESDORead, EC_StartMon, EC_StopMon, EC_SaveMon, EC_CopyMon, EC_DisconnectSlave, EC_ConnectSlave, EC_ChangeEnableSetting, IOL_ReadObj, and IOL_WriteObj.
- An error occurs in the following case. *Error* will change to TRUE.
  - A packet data save operation to an internal file in the main memory of the CPU Unit is in progress.
  - More than 32 of the following instructions were executed at the same time: EC_CoESDOWrite, EC_CoESDORead, EC_StartMon, EC_StopMon, EC_SaveMon, EC_CopyMon, EC_DisconnectSlave, EC_ConnectSlave, EC_ChangeEnableSetting, IOL_ReadObj and IOL_WriteObj.

Version Information

Depending on the unit version of the CPU Unit and the Sysmac Studio version, the following restrictions apply:

- For NX701 and NJ101 CPU Units, the instruction can be used with Sysmac Studio version 1.13 or higher.
- For an NX1P2 CPU Unit, the instruction can be used with Sysmac Studio version 1.17 or higher.
- For an NJ301 CPU Unit, the instruction can be used with the unit version 1.10 or later and Sysmac Studio version 1.12 or higher.
- For an NY-series Controller, the instruction can be used with Sysmac Studio version 1.17 or higher.

Sample Programming

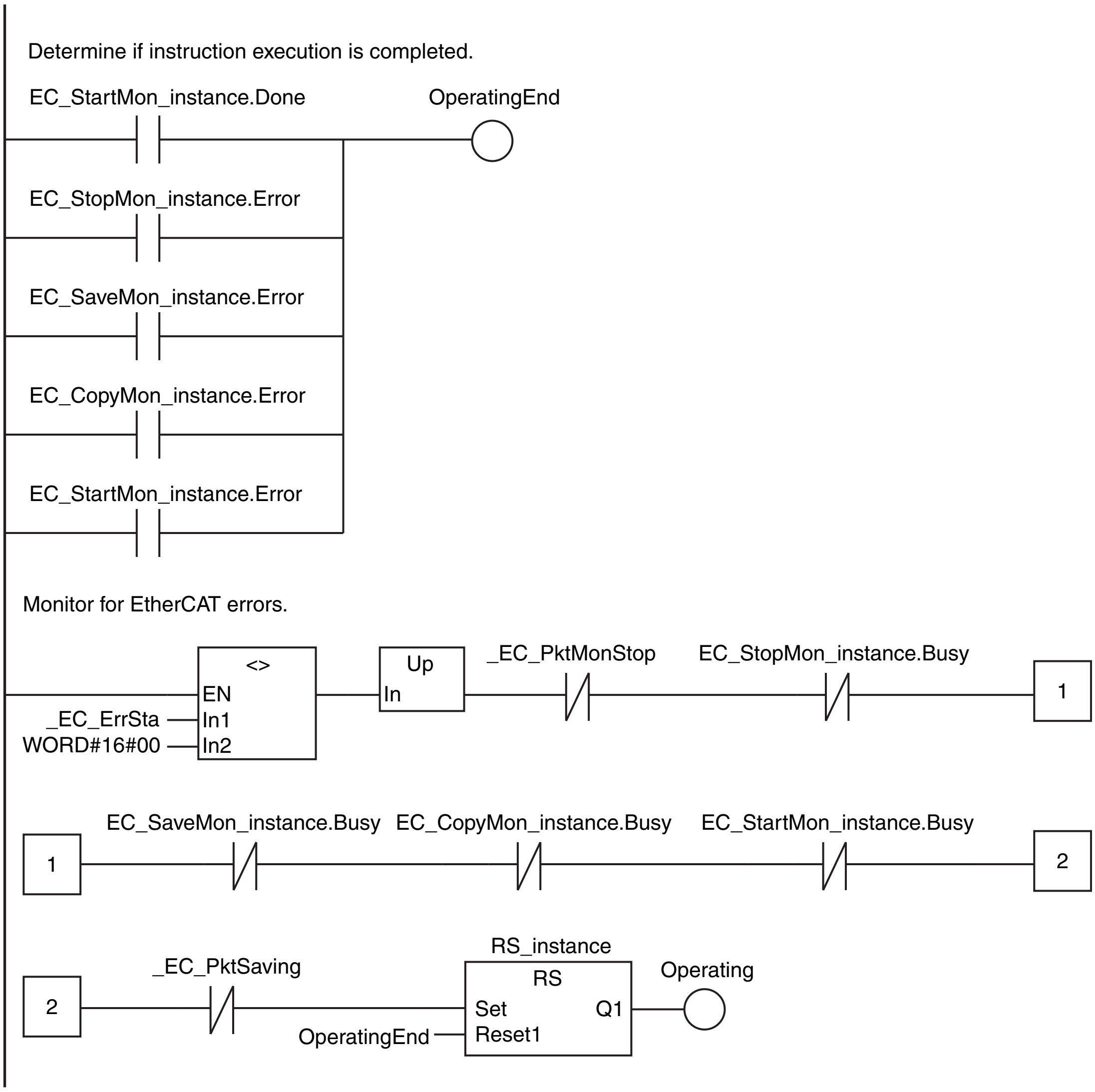
This sample transfers EtherCAT communications packets to an SD Memory Card when an EtherCAT slave error occurs. The file name is ‘PacketFile.’ The processing procedure is as follows:

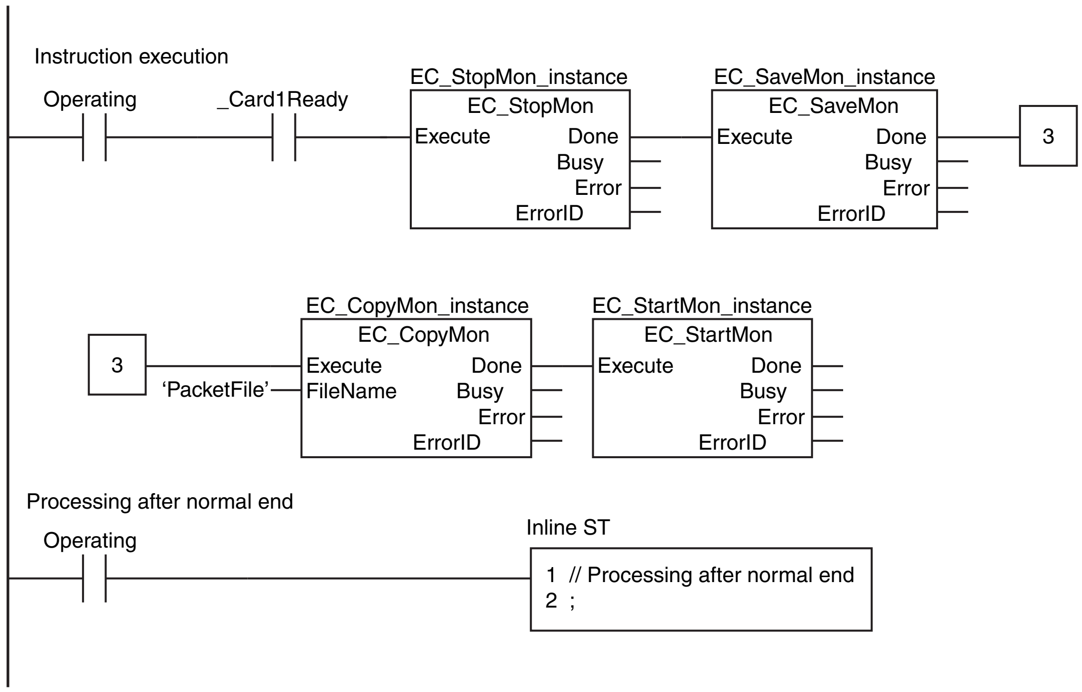
1. The system-defined variable *_EC_ErrSta* (EtherCAT Error) is monitored and processing is started if an error occurs.
2. The EC_StopMon instruction is used to stop execution of packet monitoring for EtherCAT communications.
3. The EC_SaveMon instruction is used to save EtherCAT communications packet data to an internal file in the main memory of the CPU Unit.
4. The EC_CopyMon instruction is used to copy that file to the SD Memory Card.
5. The EC_StartMon instruction is used to restart execution of packet monitoring for EtherCAT communications.

LD

Internal Variables	Variable	Data type	Initial value	Comment
	OperatingEnd	BOOL	FALSE	Processing completed
	Operating	BOOL	FALSE	Execution condition
	RS_instance	RS		
	EC_StopMon_instance	EC_StopMon		
	EC_SaveMon_instance	EC_SaveMon		
	EC_CopyMon_instance	EC_CopyMon		
	EC_StartMon_instance	EC_StartMon		

External Variables	Variable	Data type	Constant	Comment
	_EC_ErrSta	WORD	✔	Built-in EtherCAT Error
	_EC_PktMonStop	BOOL	✔	Packet Monitoring Stopped
	_EC_PktSaving	BOOL	✔	Saving Packet Data File
	_Card1Ready	BOOL	✔	SD Memory Card Ready Flag




Instruction execution
Operating
_Card1Ready
EC_StopMon_instance
EC_StopMon
Execute
Done
Busy
Error
ErrorID
EC_SaveMon_instance
EC_SaveMon
Execute
Done
Busy
Error
ErrorID
3
3
'PacketFile'
EC_CopyMon_instance
EC_CopyMon
Execute
FileName
Done
Busy
Error
ErrorID
EC_StartMon_instance
EC_StartMon
Execute
Done
Busy
Error
ErrorID
Processing after normal end
Operating
Inline ST
1 // Processing after normal end
2 ;

ST

Internal Variables	Variable	Data type	Initial value	Comment
	EC_Err	BOOL	FALSE	Controller error in the EtherCAT Master Function Module.
	EC_Err_Trigger	BOOL	FALSE	Detect when *EC_Err* changes to TRUE.
	DoEC_PktSave	BOOL	FALSE	Processing
	Stage	INT	0	Stage change
	R_TRIG_instance	R_TRIG		
	EC_StopMon_instance	EC_StopMon		
	EC_SaveMon_instance	EC_SaveMon		
	EC_CopyMon_instance	EC_CopyMon		
	EC_StartMon_instance	EC_StartMon		

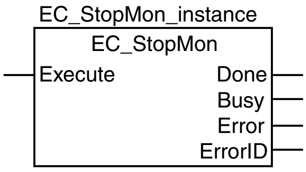
External Variables	Variable	Data type	Constant	Comment
	_EC_ErrSta	WORD	✔	Built-in EtherCAT Error
	_EC_PktMonStop	BOOL	✔	Packet Monitoring Stopped
	_EC_PktSaving	BOOL	✔	Saving Packet Data File
	_Card1Ready	BOOL	✔	SD Memory Card Ready Flag

```
// Start sequence when _EC_ErrSta changes to TRUE.
EC_Err:=(_EC_ErrSta <> WORD#16#00);
R_TRIG_instance(Clk:=EC_Err, Q=>EC_Err_Trigger);

IF ( (EC_Err_Trigger=TRUE) AND (DoEC_PktSave=FALSE) AND (_EC_PktMonStop=FALSE)
    AND (_EC_PktSaving=FALSE) AND (_Card1Ready=TRUE) ) THEN
    DoEC_PktSave:=TRUE;
    Stage       :=INT#1;
    EC_StopMon_instance(Execute:=FALSE);   // Initialize instance.
    EC_SaveMon_instance(Execute:=FALSE);
    EC_CopyMon_instance(Execute:=FALSE);
    EC_StartMon_instance(Execute:=FALSE);
END_IF;

// Instruction execution
IF (DoEC_PktSave=TRUE) THEN
    CASE Stage OF
    1 :   // Stop EtherCAT packet monitor.
        EC_StopMon_instance(
            Execute     :=TRUE);

        IF (EC_StopMon_instance.Done=TRUE) THEN
            Stage:=INT#2;   // Normal end
        ELSIF (EC_StopMon_instance.Error=TRUE) THEN
            Stage:=INT#10;  // Error end
        END_IF;

    2 :   // Save EtherCAT packet data in an internal file.
        EC_SaveMon_instance(
            Execute :=TRUE);

        IF (EC_SaveMon_instance.Done=TRUE) THEN
            Stage:=INT#3;   // Normal end
        ELSIF (EC_SaveMon_instance.Error=TRUE) THEN
            Stage:=INT#20;  // Error end
        END_IF;
```

```
    3 :   // Copy EtherCAT packet data file to the SD Memory Card.
        EC_CopyMon_instance(
            Execute  :=TRUE,
            FileName :='PacketFile');

        IF (EC_CopyMon_instance.Done=TRUE) THEN
            Stage:=INT#4;   // Normal end
        ELSIF (EC_CopyMon_instance.Error=TRUE) THEN
            Stage:=INT#30;  // Error end
        END_IF;

    4 :   // Restart EtherCAT packet monitor.
        EC_StartMon_instance(
            Execute  :=TRUE);

        IF (EC_StartMon_instance.Done=TRUE) THEN
            Stage:=INT#0;   // Normal end
        ELSIF (EC_StartMon_instance.Error=TRUE) THEN
            Stage:=INT#40;  // Error end
        END_IF;

    0 :   // Processing after normal end
        DoEC_PktSave:=FALSE;

    ELSE  // Processing after error end
        DoEC_PktSave:=FALSE;
    END_CASE;
END_IF;
```

EC_StopMon

The EC_StopMon instruction stops execution of packet monitoring for EtherCAT communications.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
EC_StopMon	Stop EtherCAT Packet Monitor	FB		EC_StopMon_instance(Execute, Done, Busy, Error, ErrorID);

Variables

Only common variables are used.

Function

The EC_StopMon instruction stops execution of packet monitoring for EtherCAT communications. The packet monitor function collects a specified number of the most recent EtherCAT communications packets.

Related System-defined Variables

Name	Meaning	Data type	Description
_EC_PktMonStop	Packet Monitoring Stopped	BOOL	This variable shows if packet monitoring is stopped. TRUE: Stopped. FALSE: Not stopped.
_EC_PktSaving	Saving Packet Data File	BOOL	This variable shows if the instruction is saving packet data in an internal file in the main memory of the CPU Unit. TRUE: Saving. FALSE: Not saving.

Additional Information

- Do the following to save collected packet data in an internal file in the main memory of the CPU Unit: First, stop packet monitoring. Then execute the EC_SaveMon instruction to save the packets.
- Refer to the *NJ/NX-series CPU Unit Built-in EtherCAT Port User's Manual* (Cat. No. W505) or *NY-series Industrial Panel PC / Industrial Box PC Built-in EtherCAT Port User's Manual* (Cat. No. W562) for details on EtherCAT communications.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.

- This instruction can be used only for the NJ/NX-series and NY-series Controller EtherCAT ports.
- You can execute a maximum of 32 of the following instructions at the same time: EC_CoESDOWrite, EC_CoESDORead, EC_StartMon, EC_StopMon, EC_SaveMon, EC_CopyMon, EC_DisconnectSlave, EC_ConnectSlave, EC_ChangeEnableSetting, IOL_ReadObj, and IOL_Write-Obj.
- An error occurs in the following case. *Error* will change to TRUE.
  - Packet monitoring is already stopped.
  - More than 32 of the following instructions were executed at the same time: EC_CoESDOWrite, EC_CoESDORead, EC_StartMon, EC_StopMon, EC_SaveMon, EC_Copy-Mon, EC_DisconnectSlave, EC_ConnectSlave, EC_ChangeEnableSetting, IOL_ReadObj and IOL_WriteObj.

Version Information

Depending on the unit version of the CPU Unit and the Sysmac Studio version, the following restrictions apply:

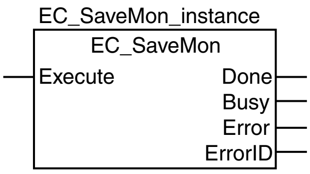
- For NX701 and NJ101 CPU Units, the instruction can be used with Sysmac Studio version 1.13 or higher.
- For an NX1P2 CPU Unit, the instruction can be used with Sysmac Studio version 1.17 or higher.
- For an NJ301 CPU Unit, the instruction can be used with the unit version 1.10 or later and Sysmac Studio version 1.12 or higher.
- For an NY-series Controller, the instruction can be used with Sysmac Studio version 1.17 or higher.

Sample Programming

Refer to the sample programming that is provided for the EC_StartMon instruction (page 2-916).

EC_SaveMon

The EC_SaveMon instruction saves EtherCAT communications packet data to an internal file in the main memory of the CPU Unit.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
EC_SaveMon	Save EtherCAT Packets	FB		EC_SaveMon_instance(Execute, Done, Busy, Error, ErrorID);

Variables

Only common variables are used.

Function

The EC_SaveMon instruction saves EtherCAT communications packet data that was collected by the packet monitoring function to an internal file in the main memory of the CPU Unit. The packet monitor function collects a specified number of the most recent EtherCAT communications packets.

Related System-defined Variables

Name	Meaning	Data type	Description
_EC_PktMonStop	Packet Monitoring Stopped	BOOL	This variable shows if packet monitoring is stopped. TRUE: Stopped. FALSE: Not stopped.
_EC_PktSaving	Saving Packet Data File	BOOL	This variable shows if the instruction is saving packet data in an internal file in the main memory of the CPU Unit. TRUE: Saving. FALSE: Not saving.

Additional Information

- You cannot execute packet monitoring while this instruction is in execution.
- Refer to the *NJ/NX-series CPU Unit Built-in EtherCAT Port User's Manual* (Cat. No. W505) or *NY-series Industrial Panel PC / Industrial Box PC Built-in EtherCAT Port User's Manual* (Cat. No. W562) for details on EtherCAT communications.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.

- This instruction can be used only for the NJ/NX-series and NY-series Controller EtherCAT ports.
- You cannot execute this instruction while packet monitoring is in progress. Execute the EC_StopMon instruction in advance to stop packet monitoring.
- You can execute a maximum of 32 of the following instructions at the same time: EC_CoESDOWrite, EC_CoESDORead, EC_StartMon, EC_StopMon, EC_SaveMon, EC_CopyMon, EC_DisconnectSlave, EC_ConnectSlave, EC_ChangeEnableSetting, IOL_ReadObj, and IOL_Write-Obj.
- An error occurs in the following case. *Error* will change to TRUE.
  - Packet monitoring is in progress.
  - More than 32 of the following instructions were executed at the same time: EC_CoESDOWrite, EC_CoESDORead, EC_StartMon, EC_StopMon, EC_SaveMon, EC_Copy-Mon, EC_DisconnectSlave, EC_ConnectSlave, EC_ChangeEnableSetting, IOL_ReadObj and IOL_WriteObj.

Version Information

Depending on the unit version of the CPU Unit and the Sysmac Studio version, the following restrictions apply:

- For NX701 and NJ101 CPU Units, the instruction can be used with Sysmac Studio version 1.13 or higher.
- For an NX1P2 CPU Unit, the instruction can be used with Sysmac Studio version 1.17 or higher.
- For an NJ301 CPU Unit, the instruction can be used with the unit version 1.10 or later and Sysmac Studio version 1.12 or higher.
- For an NY-series Controller, the instruction can be used with Sysmac Studio version 1.17 or higher.

Sample Programming

Refer to the sample programming that is provided for the EC_StartMon instruction (page 2-916).

EC_CopyMon

The EC_CopyMon instruction transfers packet data in an internal file in the main memory of the CPU Unit to a SD Memory Card.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
EC_CopyMon	Transfer EtherCAT Packets	FB	EC_CopyMon_instance EC_CopyMon Execute — Done FileName — Busy Error ErrorID	EC_CopyMon_instance(Execute, FileName, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
FileName	File name	Input	File name on the SD Memory Card	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileName																				OK

Function

The EC_CopyMon instruction transfers packet data in an internal file in the main memory of the CPU Unit to a SD Memory Card. *FileName* specifies the file name on the SD Memory Card.

Related System-defined Variables

Name	Meaning	Data type	Description
_EC_PktSaving	Saving Packet Data File	BOOL	This variable shows if the instruction is saving packet data in an internal file in the main memory of the CPU Unit. TRUE: Saving. FALSE: Not saving.

Additional Information

Refer to the *NJ/NX-series CPU Unit Built-in EtherCAT Port User's Manual* (Cat. No. W505) or *NY-series Industrial Panel PC / Industrial Box PC Built-in EtherCAT Port User's Manual* (Cat. No. W562) for details on EtherCAT communications.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the NJ/NX-series and NY-series Controller EtherCAT ports.
- You cannot execute this instruction while a packet save operation is in progress.
- To use this instruction, execute the EC_SaveMon instruction in advance to save the packet data in an internal file in the main memory of the CPU Unit.
- You can execute a maximum of 32 of the following instructions at the same time: EC_CoESDOWrite, EC_CoESDORead, EC_StartMon, EC_StopMon, EC_SaveMon, EC_CopyMon, EC_DisconnectSlave, EC_ConnectSlave, EC_ChangeEnableSetting, IOL_ReadObj, and IOL_WriteObj.
- An error occurs in the following case. *Error* will change to TRUE.
  - A packet data file save operation is in progress.
  - More than 32 of the following instructions were executed at the same time: EC_CoESDOWrite, EC_CoESDORead, EC_StartMon, EC_StopMon, EC_SaveMon, EC_CopyMon, EC_DisconnectSlave, EC_ConnectSlave, EC_ChangeEnableSetting, IOL_ReadObj and IOL_WriteObj.

Version Information

Depending on the unit version of the CPU Unit and the Sysmac Studio version, the following restrictions apply:

- For NX701 and NJ101 CPU Units, the instruction can be used with Sysmac Studio version 1.13 or higher.
- For an NX1P2 CPU Unit, the instruction can be used with Sysmac Studio version 1.17 or higher.
- For an NJ301 CPU Unit, the instruction can be used with the unit version 1.10 or later and Sysmac Studio version 1.12 or higher.
- For an NY-series Controller, the instruction can be used with Sysmac Studio version 1.17 or higher.

Sample Programming

Refer to the sample programming that is provided for the EC_StartMon instruction (page 2-916).

EC_DisconnectSlave

The EC_DisconnectSlave instruction disconnects the specified slave from the network.

Instruction	Name	FB/FUN	Graphic expression	ST expression
EC_Discon-nectSlave	Disconnect EtherCAT Slave	FB	EC_DisconnectSlave_instance EC_DisconnectSlave Execute — Done NodeAdr — Busy Error ErrorID	EC_DisconnectSlave_instance(Execute, NodeAdr, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
NodeAdr	Slave node address	Input	Node address of the slave to disconnect	1 to 512*	---	---

* The range is 1 to 192 for an NJ-series CPU Unit.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
NodeAdr							OK													

Function

The EC_DisconnectSlave instruction disconnects the slave specified with slave node address *NodeAdr* from the EtherCAT network.
Here, disconnection from the network means that the slave is placed in a state in which it does not operate even though it still exists on the network.

Related System-defined Variables

Name	Meaning	Data type	Description
_EC_EntrySlavTbl[i] “i” is the node address.	Network Connected Slave Table	BOOL[]	This variable shows if slaves are part of (i.e., exist on) the network. TRUE: Part of the network. FALSE: Not part of the network.
_EC_DisconnSlavTbl[i] “i” is the node address.	Disconnected Slave Table	BOOL[]	This variable shows the slaves for which there are currently disconnect commands in effect. TRUE: Disconnect command is in effect. FALSE: Disconnect command is not in effect.
_EC_DisableSlavTbl[i] “i” is the node address.	Disabled Slave Table	BOOL[]	This variable shows if slaves are disabled on the network. TRUE: Disabled. FALSE: Not disabled.

Additional Information

Refer to the *NJ/NX-series CPU Unit Built-in EtherCAT Port User's Manual* (Cat. No. W505) or *NY-series Industrial Panel PC / Industrial Box PC Built-in EtherCAT Port User's Manual* (Cat. No. W562) for details on EtherCAT communications.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the NJ/NX-series and NY-series Controller EtherCAT ports.
- If there are slaves with daisy-chain connections (i.e., connected to the output port) after the disconnected slave, they are disconnected from the EtherCAT network also.
- You cannot execute this instruction during execution of the following instructions: EC_DisconnectSlave, EC_ConnectSlave, EC_ChangeEnableSetting, ResetECError, RestartNXUnit, and NX_ChangeWriteMode.
- You can execute a maximum of 32 of the following instructions at the same time: EC_CoESDOWrite, EC_CoESDORead, EC_StartMon, EC_StopMon, EC_SaveMon, EC_CopyMon, EC_DisconnectSlave, EC_ConnectSlave, EC_ChangeEnableSetting, IOL_ReadObj, and IOL_WriteObj.
- An error occurs in the following case. *Error* will change to TRUE.
  - The slave specified with *NodeAdr* is not part of the EtherCAT network. That is, the value of *_EC_EntrySlavTbl[i]* (Network Connected Slave Table) is FALSE.
  - The slave specified with *NodeAdr* is disabled.
  - The EC_DisconnectSlave, EC_ConnectSlave, EC_ChangeEnableSetting, ResetECError, RestartNXUnit, or NX_ChangeWriteMode instruction is already in execution.
  - More than 32 of the following instructions were executed at the same time: EC_CoESDOWrite, EC_CoESDORead, EC_StartMon, EC_StopMon, EC_SaveMon, EC_CopyMon, EC_DisconnectSlave, EC_ConnectSlave, EC_ChangeEnableSetting, IOL_ReadObj and IOL_WriteObj.

Sample Programming

This sample disconnects slave 1 from the EtherCAT network and then connects it again. When *Trigger1* changes to TRUE, the EC_DisconnectSlave instruction is executed to disconnect slave 1. When *Trigger2* changes to TRUE, the EC_ConnectSlave instruction is executed to connect slave 1 again.

Exclusive Control of Instructions

You cannot execute the EC_DisconnectSlave and EC_ConnectSlave instructions at the same time. Both of these instructions are executed over more than one task. Confirm the completion of the instruction that was executed first before you execute the other instruction. The *ExclusiveFlg* variable (Instruction Exclusive Flag) is used for this purpose. If the value of *ExclusiveFlg* is TRUE, then one of the instructions is in execution. Do not execute the next instruction while the value of *ExclusiveFlg* is TRUE.

You cannot execute the EC_DisconnectSlave and EC_ConnectSlave instructions at the same time even in separate tasks. Therefore, *ExclusiveFlg* is defined as a global variable in this sample programming. That allows this sample programming to perform exclusive control with instructions in other tasks. The same global variable, *ExclusiveFlg*, must also be used in the other tasks to perform exclusive control of the instructions.

You cannot execute the EC_ChangeEnableSetting instruction at the same time as the EC_Disconnect-Slave or EC_ConnectSlave instruction. The sample programming that is provided for the EC_Chan-geEnableSetting instruction on page 2-937 uses the same *ExclusiveFlg* global variable as in this sample programming as an example of exclusive control of instructions.

Definitions of Global Variables

Global Variables

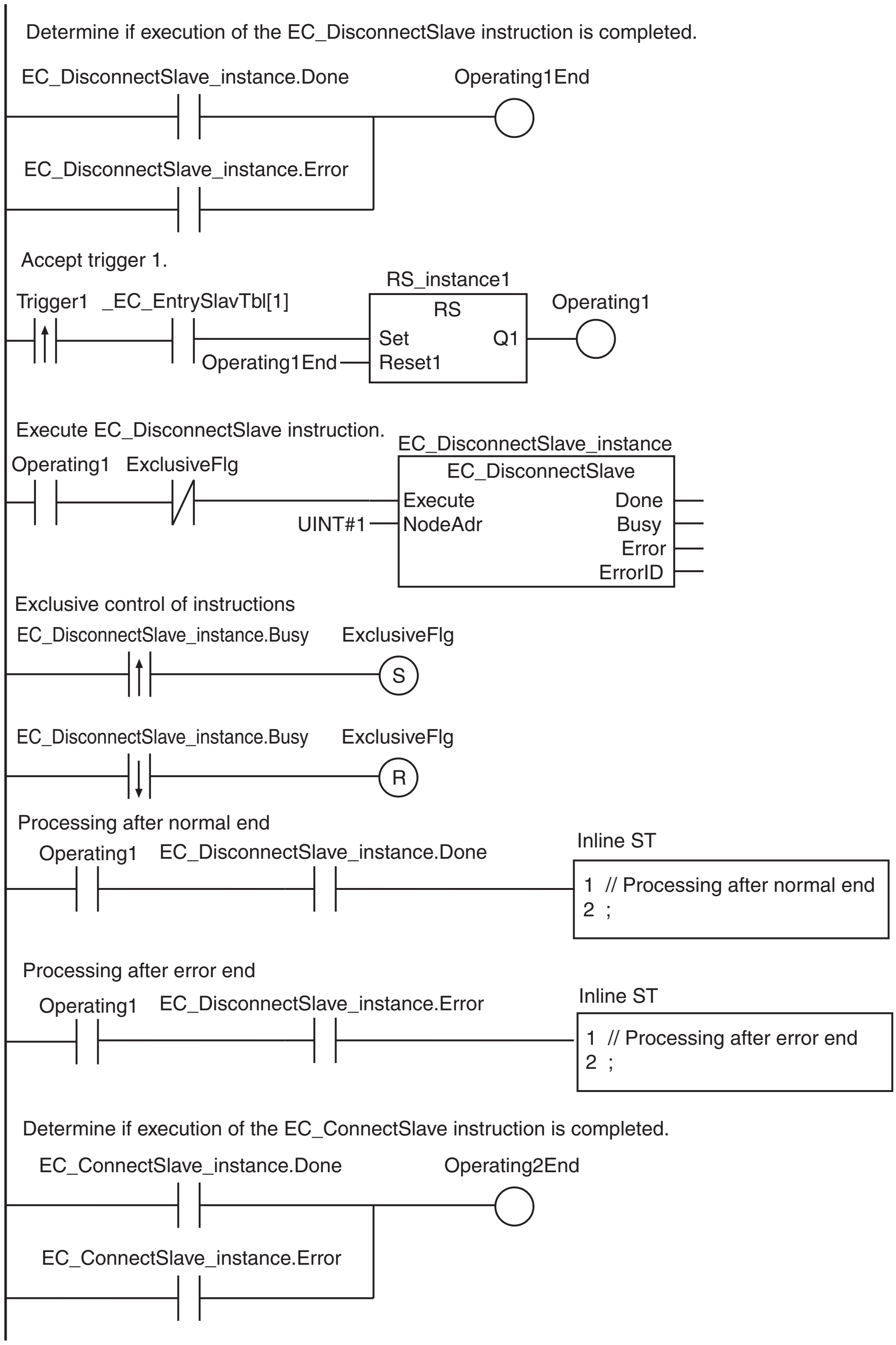
Variable	Data type	Initial value	Comment
ExclusiveFlg	BOOL	FALSE	Instruction Exclusive Flag

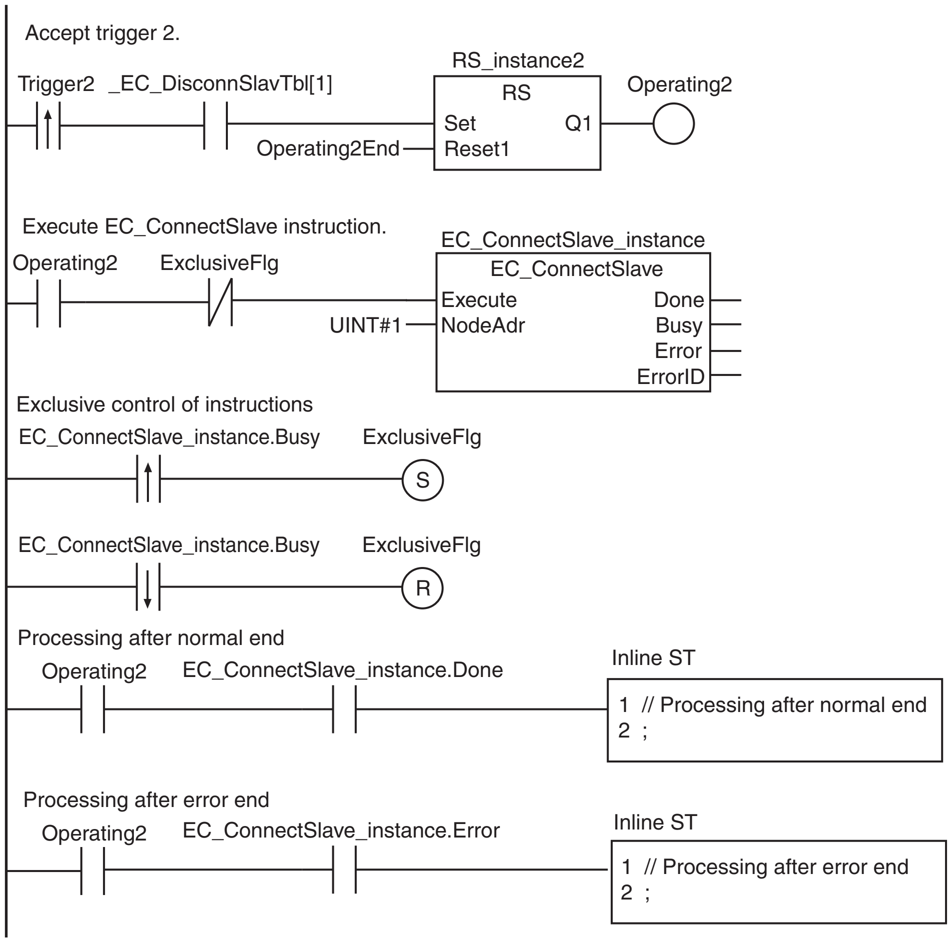
LD

Internal Variables	Variable	Data type	Initial value	Comment
	Operating1End	BOOL	FALSE	Processing 1 completed.
	Trigger1	BOOL	FALSE	Execution condition 1
	Operating1	BOOL	FALSE	Processing 1
	RS_instance1	RS		
	EC_DisconnectSlave_instance	EC_DisconnectSlave		
	Operating2End	BOOL	FALSE	Processing 2 completed.
	Trigger2	BOOL	FALSE	Execution condition 2
	Operating2	BOOL	FALSE	Processing 2
	RS_instance2	RS		
	EC_ConnectSlave_instance	EC_ConnectSlave		

External Variables	Variable	Data type	Constant	Comment
	_EC_EntrySlavTbl	ARRAY[1..512] OF BOOL *1	✔	Network Connected Slave Table
	_EC_DisconnSlavTbl	ARRAY[1..512] OF BOOL *1	✔	Disconnected Slave Table
	ExclusiveFlg	BOOL	---	Instruction Exclusive Flag

*1 The data type is ARRAY [1..192] OF BOOL for an NJ-series CPU Unit.


Determine if execution of the EC_DisconnectSlave instruction is completed.
EC_DisconnectSlave_instance.Done
Operating1End
EC_DisconnectSlave_instance.Error
Accept trigger 1.
RS_instance1
Trigger1
_EC_EntrySlavTbl[1]
RS
Operating1
Set
Q1
Operating1End
Reset1
Execute EC_DisconnectSlave instruction.
EC_DisconnectSlave_instance
Operating1
ExclusiveFlg
EC_DisconnectSlave
Execute
Done
UINT#1
NodeAdr
Busy
Error
ErrorID
Exclusive control of instructions
EC_DisconnectSlave_instance.Busy
ExclusiveFlg
S
EC_DisconnectSlave_instance.Busy
ExclusiveFlg
R
Processing after normal end
Operating1
EC_DisconnectSlave_instance.Done
Inline ST
1 // Processing after normal end
2 ;
Processing after error end
Operating1
EC_DisconnectSlave_instance.Error
Inline ST
1 // Processing after error end
2 ;
Determine if execution of the EC_ConnectSlave instruction is completed.
EC_ConnectSlave_instance.Done
Operating2End
EC_ConnectSlave_instance.Error


Accept trigger 2.
RS_instance2
Trigger2
_EC_DisconnSlavTbl[1]
RS
Operating2
Set
Q1
Operating2End
Reset1
Execute EC_ConnectSlave instruction.
EC_ConnectSlave_instance
Operating2
ExclusiveFlg
EC_ConnectSlave
Execute
Done
UINT#1
NodeAdr
Busy
Error
ErrorID
Exclusive control of instructions
EC_ConnectSlave_instance.Busy
ExclusiveFlg
S
EC_ConnectSlave_instance.Busy
ExclusiveFlg
R
Processing after normal end
Inline ST
Operating2
EC_ConnectSlave_instance.Done
1 // Processing after normal end
2 ;
Processing after error end
Inline ST
Operating2
EC_ConnectSlave_instance.Error
1 // Processing after error end
2 ;

ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger1	BOOL	FALSE	Execution condition 1
	LastTrigger1	BOOL	FALSE	Value of *Trigger1* from previous task period
	Operating1Start	BOOL	FALSE	Processing 1 started.
	Operating1	BOOL	FALSE	Processing 1
	DisconnectSet	BOOL	FALSE	EC_DisconnectSlave instruction execution is in progress.
	DisconnectReset	BOOL	FALSE	EC_DisconnectSlave instruction execution is not in progress.
	EC_DisconnectSlave_instance	EC_DisconnectSlave		
	Trigger2	BOOL	FALSE	Execution condition 2
	LastTrigger2	BOOL	FALSE	Value of *Trigger2* from previous task period
	Operating2Start	BOOL	FALSE	Processing 2 started.
	Operating2	BOOL	FALSE	Processing 2
	ConnectSet	BOOL	FALSE	EC_ConnectSlave instruction execution is in progress.
	ConnectReset	BOOL	FALSE	EC_ConnectSlave instruction execution is not in progress.
	EC_ConnectSlave_instance	EC_ConnectSlave		
	R_TRIG_instance1	R_TRIG		
	F_TRIG_instance1	F_TRIG		
	RS_instance1	RS		
	R_TRIG_instance2	R_TRIG		
	F_TRIG_instance2	F_TRIG		
	RS_instance2	RS		

External Variables	Variable	Data type	Constant	Comment
	_EC_EntrySlavTbl	ARRAY[1..512] OF BOOL *1	✔	Network Connected Slave Table
	_EC_DisconnSlavTbl	ARRAY[1..512] OF BOOL *1	✔	Disconnected Slave Table
	ExclusiveFlg	BOOL	---	Instruction Exclusive Flag

*1 The data type is ARRAY [1..192] OF BOOL for an NJ-series CPU Unit.

```
// Detect when Trigger1 changes to TRUE.
IF ( (Trigger1=TRUE) AND (LastTrigger1=FALSE) AND (_EC_EntrySlavTbl[1]=TRUE) ) THEN
    Operating1Start:=TRUE;
    Operating1     :=TRUE;
END_IF;
LastTrigger1:=Trigger1;

// Initialize EC_DisconnectSlave instruction.
IF (Operating1Start=TRUE) THEN
    EC_DisconnectSlave_instance(Execute:=FALSE);
    Operating1Start:=FALSE;
END_IF;

// Execute EC_DisconnectSlave instruction.
IF (Operating1=TRUE) THEN
```

```
    EC_DisconnectSlave_instance(
       Execute:=NOT(ExclusiveFlg),
       NodeAdr:=UINT#1);

    // Exclusive control of instructions
    R_TRIG_instance1(EC_DisconnectSlave_instance.Busy, DisconnectSet);
    F_TRIG_instance1(EC_DisconnectSlave_instance.Busy, DisconnectReset);
    RS_instance1(DisconnectSet, DisconnectReset, ExclusiveFlg);

    IF (EC_DisconnectSlave_instance.Done=TRUE) THEN
        // Processing after normal end
        Operating1:=FALSE;
    END_IF;

    IF (EC_DisconnectSlave_instance.Error=TRUE) THEN
        // Processing after error end
        Operating1:=FALSE;
    END_IF;
END_IF;

// Detect when Trigger2 changes to TRUE.
IF ( (Trigger2=TRUE) AND (LastTrigger2=FALSE) AND (_EC_DisconnSlavTbl[1]=TRUE) ) THEN
    Operating2Start:=TRUE;
    Operating2     :=TRUE;
END_IF;
LastTrigger2:=Trigger2;

// Initialize EC_ConnectSlave instruction.
IF (Operating2Start=TRUE) THEN
    EC_ConnectSlave_instance(Execute:=FALSE);
    Operating2Start:=FALSE;
END_IF;

// Execute EC_ConnectSlave instruction.
IF (Operating2=TRUE) THEN
    EC_ConnectSlave_instance(
        Execute:=NOT(ExclusiveFlg),
        NodeAdr:=UINT#1);

    // Exclusive control of instructions
    R_TRIG_instance2(EC_ConnectSlave_instance.Busy, ConnectSet);
    F_TRIG_instance2(EC_ConnectSlave_instance.Busy, ConnectReset);
    RS_instance2(ConnectSet, ConnectReset, ExclusiveFlg);

    IF (EC_ConnectSlave_instance.Done=TRUE) THEN
            // Processing after normal end
            Operating2:=FALSE;
    END_IF;

    IF (EC_ConnectSlave_instance.Error=TRUE) THEN
            // Processing after error end
            Operating2:=FALSE;
    END_IF;
END_IF;
```

EC_ConnectSlave

The EC_ConnectSlave instruction connects the specified slave to the EtherCAT network.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
EC_Connect-Slave	Connect Ether-CAT Slave	FB	EC_ConnectSlave_instance EC_ConnectSlave Execute — Done NodeAdr — Busy Error ErrorID	EC_ConnectSlave_instance(Execute, NodeAdr, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
NodeAdr	Slave node address	Input	Node address of the slave to connect	0*1 to 512*2	---	---

*1 Here, 0 means all of the slaves that are registered in the network settings.

*2 The range is 1 to 192 for an NJ-series CPU Unit.

Version Information

For an NJ-series CPU Unit, the valid range of slave node addresses depends on the version as follows:

- Version 1.10 or later: 0 to 192
- Version 1.09 or earlier: 1 to 192

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
NodeAdr							OK													

Function

The EC_ConnectSlave instruction connects the slave specified with slave node address *NodeAdr* to the EtherCAT network.

Here, connection to the network means that the slave exists on the network and it is placed in a state in which it operates.

Related System-defined Variables

Name	Meaning	Data type	Description
_EC_EntrySlavTbl[i] “i” is the node address.	Network Connected Slave Table	BOOL[]	This variable shows if slaves are part of (i.e., exist on) the network. TRUE: Part of the network. FALSE: Not part of the network.
_EC_DisconnSlavTbl[i] “i” is the node address.	Disconnected Slave Table	BOOL[]	This variable shows the slaves for which there are currently disconnect commands in effect. TRUE: Disconnect command is in effect. FALSE: Disconnect command is not in effect.

Additional Information

Refer to the *NJ/NX-series CPU Unit Built-in EtherCAT Port User’s Manual* (Cat. No. W505) or *NY-series Industrial Panel PC / Industrial Box PC Built-in EtherCAT Port User’s Manual* (Cat. No. W562) for details on EtherCAT communications.

Precautions for Correct Use

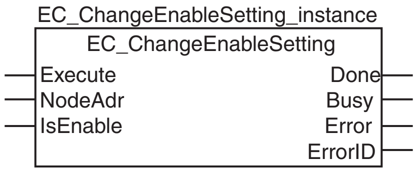
- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the NJ/NX-series and NY-series Controller EtherCAT ports.
- You cannot execute this instruction during execution of the following instructions: EC_Disconnect-Slave, EC_ConnectSlave, EC_ChangeEnableSetting, ResetECError, RestartNXUnit, and NX_ChangeWriteMode.
- You can execute a maximum of 32 of the following instructions at the same time: EC_CoESDOWrite, EC_CoESDORead, EC_StartMon, EC_StopMon, EC_SaveMon, EC_CopyMon, EC_DisconnectSlave, EC_ConnectSlave, EC_ChangeEnableSetting, IOL_ReadObj, and IOL_Write-Obj.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The slave specified with *NodeAdr* is not part of the EtherCAT network. That is, the value of *_EC_EntrySlavTbl[i]* (Network Connected Slave Table) is FALSE.
  - The EC_DisconnectSlave, EC_ConnectSlave, EC_ChangeEnableSetting, ResetECError, RestartNXUnit, or NX_ChangeWriteMode instruction is already in execution.
  - More than 32 of the following instructions were executed at the same time: EC_CoESDOWrite, EC_CoESDORead, EC_StartMon, EC_StopMon, EC_SaveMon, EC_Copy-Mon, EC_DisconnectSlave, EC_ConnectSlave, EC_ChangeEnableSetting, IOL_ReadObj and IOL_WriteObj.

Sample Programming

Refer to the sample programming that is provided for the EC_DisconnectSlave instruction (page 2-928).

EC_ChangeEnableSetting

The EC_ChangeEnableSetting instruction enables or disables an EtherCAT slave.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
EC_ChangeEnableSetting	Enable/Disable EtherCAT Slave	FB		EC_ChangeEnableSetting_instance(Execute, NodeAdr, IsEnable, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
NodeAdr	Slave node address	Input	Node address of the EtherCAT slave to enable or disable	1 to 512*	---	1
IsEnable	Enable/disable designation		Designation of whether to enable or disable the specified EtherCAT slave TRUE: Enable FALSE: Disable	Depends on data type.		TRUE

* The range is 1 to 192 for an NJ-series CPU Unit.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
NodeAdr							OK													
IsEnable	OK																			

Function

The EC_ChangeEnableSetting instruction enables or disables the EtherCAT slave that is specified with slave node address *NodeAdr*. The slave is enabled if enable/disable designation *IsEnable* is TRUE and disabled if it is FALSE.

Enabling or disabling the slave is completed when the instruction is completed normally (i.e., when the value of *Done* changes to TRUE).

Whether the instruction is completed normally depends on the status of the specified EtherCAT slave, i.e., whether it is enabled or disabled, whether it is participating or not participating, and whether it exists in the EtherCAT network. The status after execution of the instruction is given in the following table according to the status of the EtherCAT slave before execution of the instruction.

Status before instruction execution			Value of *IsEnable*	Status after instruction execution	
Enabled/disabled	**Participating/not participating**	**Exists *1**		**Normal/error end**	**Enabled/disabled**
Enabled	Participating	Yes	TRUE (Enabled)	Normal end	Enabled
	Not participating	Yes		Error end *2	Enabled
		No			
Disabled	--- *3	Yes		Normal end	Enabled
		No		Error end *4	Disabled *4
Enabled	Participating	Yes	FALSE (Disabled)	Normal end	Disabled
	Not participating	Yes		Error end *2	Enabled
		No			
Disabled	--- *3	Yes		Normal end	Disabled
		No		Error end *4	Disabled *4

*1. This indicates whether the specified EtherCAT slave is physically connected to the EtherCAT network. Yes: Physically connected. No: Not physically connected.

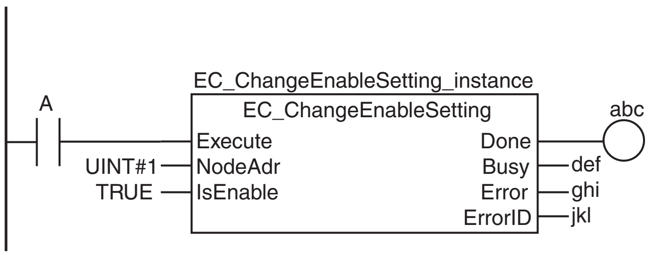
*2. For a CPU Unit with unit version 1.30 or later, Error code 180A is returned. For a CPU Unit with unit version earlier than 1.30, Error code 1800 is returned.

*3. There is no participating/not participating distinction for EtherCAT slaves that are disabled.

*4. The normal/error end status is error end, the enabled/disabled status before the instruction execution is retained, and Error code 1801 is returned.

The following example shows how to enable the EtherCAT slave at node address 1. UINT#1 is specified for *NodeAdr* and TRUE is specified for *IsEnable*.

LD



ST

```
EC_ChangeEnableSetting_instance(A, UINT#1, TRUE,
    abc, def, ghi, jkl);
```

Related System-defined Variables

Name	Meaning	Data type	Description
_EC_EntrySlavTbl [i] "i" is the node address.	Network Connected Slave Table	BOOL[]	This variable shows if slaves are part of (i.e., exist on) the network. TRUE: Part of the network. FALSE: Not part of the network.
_EC_DisconnSlavTbl[i] "i" is the node address.	Disconnected Slave Table	BOOL[]	This variable shows the slaves for which there are currently disconnect commands in effect. TRUE: Disconnect command is in effect. FALSE: Disconnect command is not in effect.
_EC_DisableSlavTbl[i] "i" is the node address.	Disabled Slave Table	BOOL[]	This variable shows if slaves are disabled. TRUE: Disabled. FALSE: Not disabled or not part of the network.

Additional Information

- Refer to the *NJ/NX-series CPU Unit Built-in EtherCAT Port User's Manual* (Cat. No. W505) or *NY-series Industrial Panel PC / Industrial Box PC Built-in EtherCAT Port User's Manual* (Cat. No. W562) for details on EtherCAT communications.
- Use the EC_ConnectSlave instruction on page 2-935 to connect an EtherCAT slave to the EtherCAT network.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the NJ/NX-series and NY-series Controller EtherCAT ports.
- You cannot execute this instruction during execution of the following instructions: EC_Disconnect-Slave, EC_ConnectSlave, EC_ChangeEnableSetting, ResetECError, RestartNXUnit, and NX_ChangeWriteMode.
- The execution results of this instruction are not saved in non-volatile memory in the CPU Unit. Therefore, if the power supply to the Controller is cycled after execution of this instruction or if the user program is downloaded, the enable/disable setting of the EtherCAT slave will return to the value that was set from the Sysmac Studio.
- You can execute a maximum of 32 of the following instructions at the same time: EC_CoESDOWrite, EC_CoESDORead, EC_StartMon, EC_StopMon, EC_SaveMon, EC_CopyMon, EC_DisconnectSlave, EC_ConnectSlave, EC_ChangeEnableSetting, IOL_ReadObj, and IOL_Write-Obj.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The slave specified with *NodeAdr* is not part of the EtherCAT network. That is, the value of *_EC_EntrySlavTbl[i]* (Network Connected Slave Table) is FALSE.
  - The value of *NodeAdr* is outside of the valid range.
  - The EC_DisconnectSlave, EC_ConnectSlave, EC_ChangeEnableSetting, ResetECError, RestartNXUnit, or NX_ChangeWriteMode instruction is already in execution.
  - More than 32 of the following instructions were executed at the same time: EC_CoESDOWrite, EC_CoESDORead, EC_StartMon, EC_StopMon, EC_SaveMon, EC_Copy-Mon, EC_DisconnectSlave, EC_ConnectSlave, EC_ChangeEnableSetting, IOL_ReadObj and IOL_WriteObj.

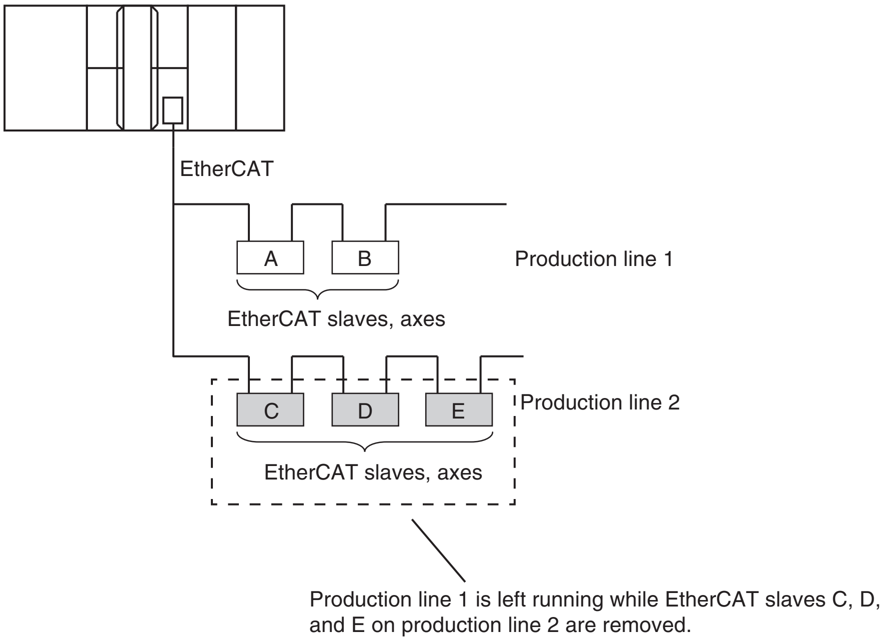
Version Information

A CPU Unit with unit version 1.04 or later and Sysmac Studio version 1.05 or higher are required to use this instruction.

Sample Programming

- **Example of Disconnecting EtherCAT Slaves from the EtherCAT Network**

Production line 1 in the following system is left running while EtherCAT slaves C, D, and E on production line 2 are removed. Motion control axes are already set for EtherCAT slaves C, D, and E. Therefore, the EtherCAT slaves are disabled and the axes are changed to unused axes.



Procedure

The operating procedure for the sample programming is as follows:

1. The operator presses a button on an HMI to turn ON the execution condition.
2. The Controller disables EtherCAT slaves C, D, and E. Also, the axes for those slaves are changed to unused axes.
3. When disabling and changing the axes to unused axes is completed for all three slaves, the Controller lights a removal OK lamp.
4. After the operator confirms that the removal OK lamp is lit, the operator removes the three EtherCAT slaves.

Instruction to Change Axes to Unused Axes

The MC_ChangeAxisUse instruction is used to change the axes to unused axes. Refer to the NJ/NX-*series Motion Control Instructions Reference Manual* (Cat. No. W508) or *NY-series Motion Control Instructions Reference Manual* (Cat. No. W561) for the detailed specifications of the MC_ChangeAxisUse instruction.

Exclusive Control of Instructions

You can execute only one EC_ChangeEnableSetting instruction at the same time. Also, the EC_ChangeEnableSetting instruction is executed over more than one task. Confirm the completion of the EC_ChangeEnableSetting instruction before you execute the next EC_ChangeEnableSetting instruction. The *ExclusiveFlg* variable (Instruction Exclusive Flag) is used for this purpose. If the value of *ExclusiveFlg* is TRUE, then an EC_ChangeEnableSetting instruction is in execution. Do not execute the next EC_ChangeEnableSetting instruction while the value of *ExclusiveFlg* is TRUE.

You cannot execute the EC_ChangeEnableSetting instruction at the same time as another EC_ChangeEnableSetting instruction is in execution in another task. Therefore, *ExclusiveFlg* is defined as a global variable in this sample programming. That allows this sample programming to perform exclusive control with EC_ChangeEnableSetting instructions in the other tasks. The same global variable, *ExclusiveFlg*, must also be used in the other tasks to perform exclusive control of the instructions.

You cannot execute the EC_ChangeEnableSetting instruction at the same time as the EC_DisconnectSlave or EC_ConnectSlave instruction. The sample programming that is provided for the EC_DisconnectSlave instruction on page 2-928 uses the same *ExclusiveFlg* global variable as in this sample programming as an example of exclusive control of instructions.

Axis Variables and Node Addresses for the EtherCAT Slaves

The axis variables that are assigned to the axes for EtherCAT slaves C, D, and E and the node addresses of the slaves are given in the following table.

EtherCAT slaves	Axis variable	Node address
C	MC_Axis000	1
D	MC_Axis001	2
E	MC_Axis002	3

Definitions of Global Variables

Global Variables

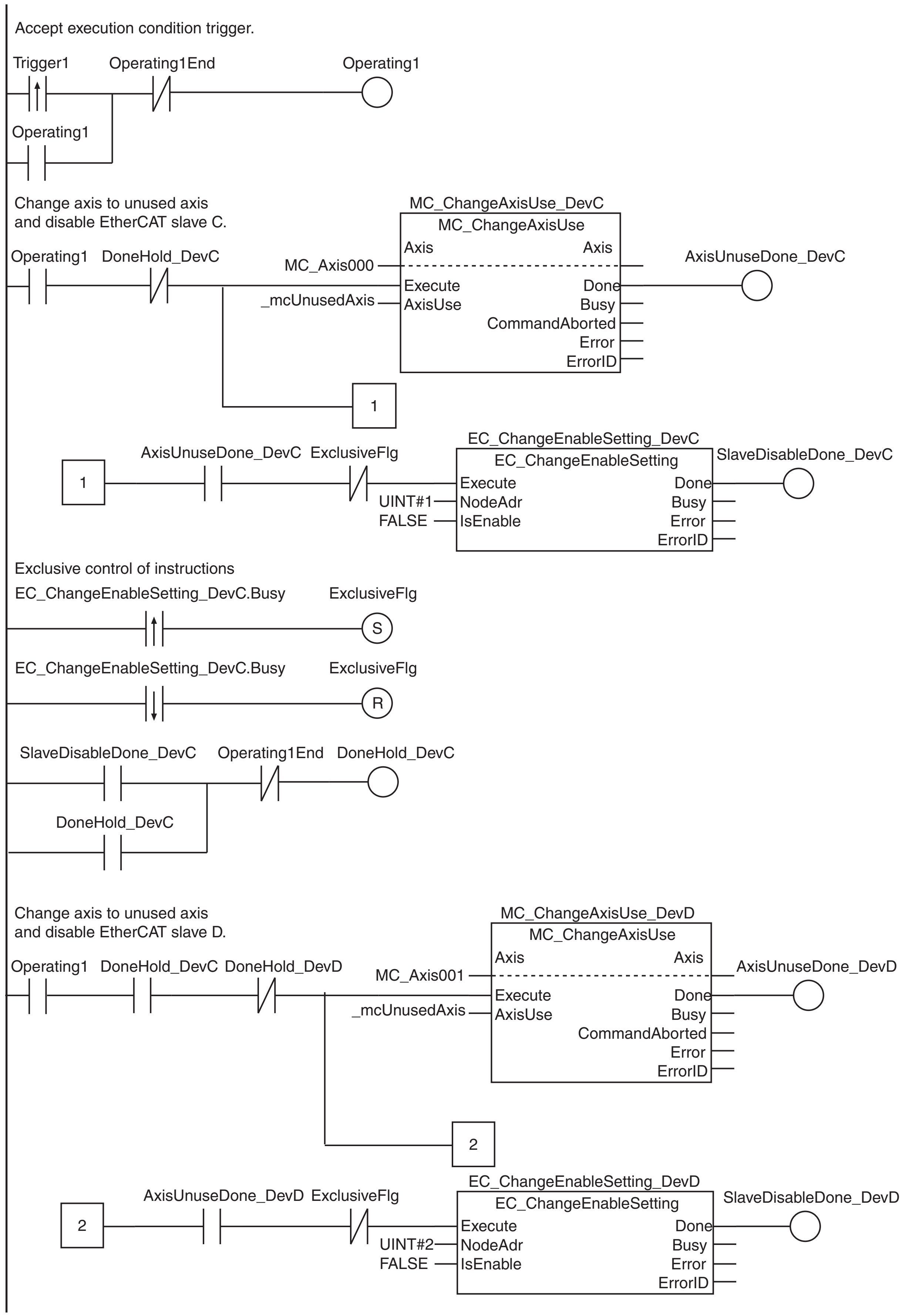
Variable	Data type	Initial value	AT specification	Constant	Comment
MC_Axis000	_sAXIS_REF		MC://_MC_AX[0]	✔	Axis variable for EtherCAT slave C
MC_Axis001	_sAXIS_REF		MC://_MC_AX[1]	✔	Axis variable for EtherCAT slave D
MC_Axis002	_sAXIS_REF		MC://_MC_AX[2]	✔	Axis variable for EtherCAT slave E
ExclusiveFlg	BOOL	FALSE		---	Instruction Exclusive Flag

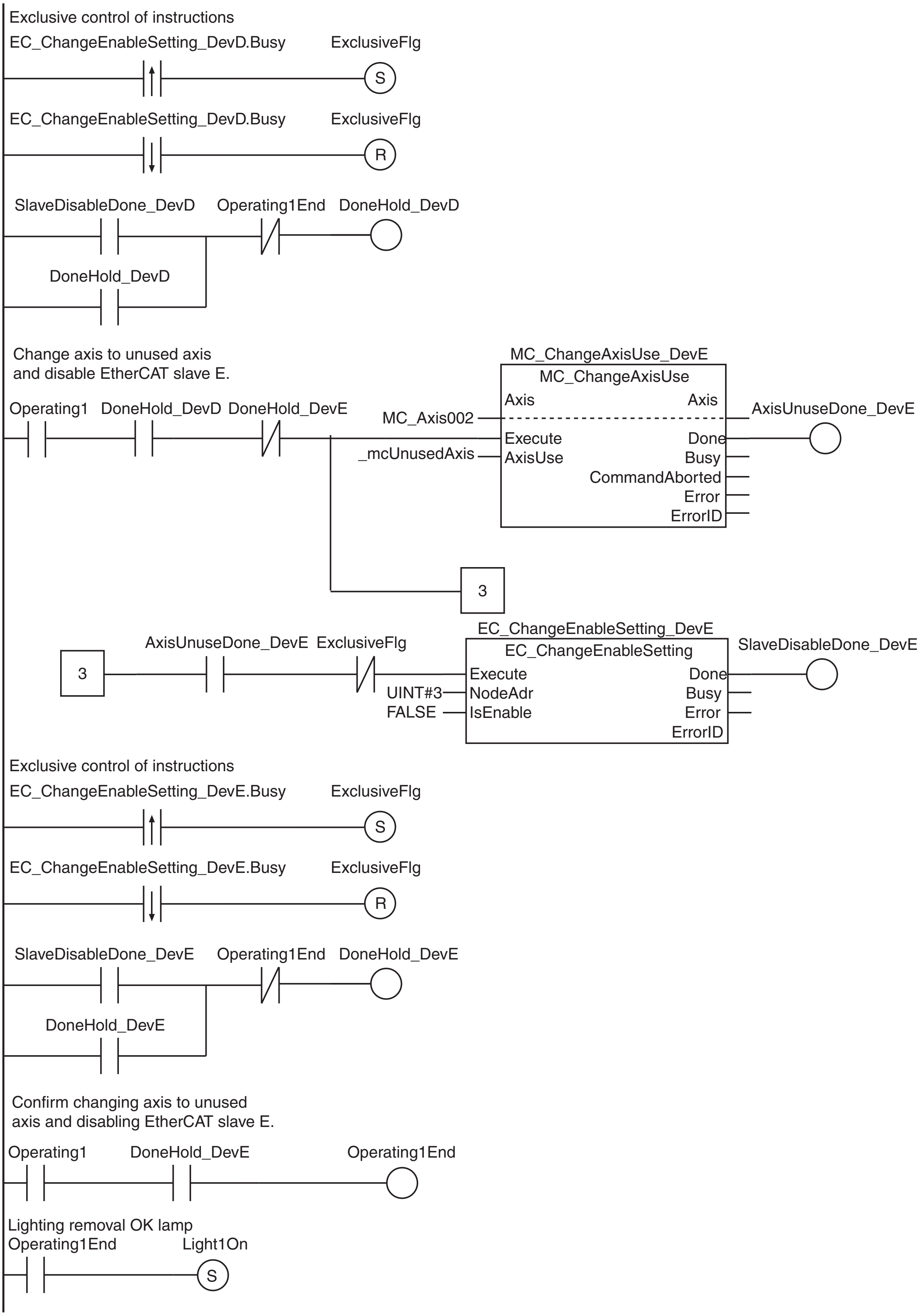
LD

Internal Variables	Variable	Data type	Initial value	Comment
	Operating1End	BOOL	FALSE	Processing completed
	Trigger1	BOOL	FALSE	Execution condition
	Operating1	BOOL	FALSE	Processing

Internal Vari-ables	Variable	Data type	Initial value	Comment
	AxisUnuseDone_DevC	BOOL	FALSE	Changing axis to unused axis completed for EtherCAT slave C
	SlaveDisableDone_DevC	BOOL	FALSE	Disabling EtherCAT slave C com-pleted
	DoneHold_DevC	BOOL	FALSE	Holding completion of processing for EtherCAT slave C
	AxisUnuseDone_DevD	BOOL	FALSE	Changing axis to unused axis completed for EtherCAT slave D
	SlaveDisableDone_DevD	BOOL	FALSE	Disabling EtherCAT slave D com-pleted
	DoneHold_DevD	BOOL	FALSE	Holding completion of processing for EtherCAT slave D
	AxisUnuseDone_DevE	BOOL	FALSE	Changing axis to unused axis completed for EtherCAT slave E
	SlaveDisableDone_DevE	BOOL	FALSE	Disabling EtherCAT slave E com-pleted
	DoneHold_DevE	BOOL	FALSE	Holding completion of processing for EtherCAT slave E
	Light1On	BOOL	FALSE	Lighting removal OK lamp
	MC_ChangeAxisUse_DevC	MC_ChangeAxisUse		
	EC_ChangeEnableSet-ting_DevC	EC_ChangeEnable-Setting		
	MC_ChangeAxisUse_DevD	MC_ChangeAxisUse		
	EC_ChangeEnableSet-ting_DevD	EC_ChangeEnable-Setting		
	MC_ChangeAxisUse_DevE	MC_ChangeAxisUse		
	EC_ChangeEnableSet-ting_DevE	EC_ChangeEnable-Setting		

External Variables	Variable	Data type	Constant	Comment
	MC_Axis000	_sAXIS_REF	✔	Axis variable for EtherCAT slave C
	MC_Axis001	_sAXIS_REF	✔	Axis variable for EtherCAT slave D
	MC_Axis002	_sAXIS_REF	✔	Axis variable for EtherCAT slave E
	ExclusiveFlg	BOOL	---	Instruction Exclusive Flag


Accept execution condition trigger.
Trigger1
Operating1End
Operating1
Operating1
Change axis to unused axis and disable EtherCAT slave C.
MC_ChangeAxisUse_DevC
MC_ChangeAxisUse
Operating1
DoneHold_DevC
MC_Axis000
Axis
Axis
AxisUnuseDone_DevC
Execute
Done
_mcUnusedAxis
AxisUse
Busy
CommandAborted
Error
ErrorID
1
EC_ChangeEnableSetting_DevC
1
AxisUnuseDone_DevC
ExclusiveFlg
EC_ChangeEnableSetting
SlaveDisableDone_DevC
Execute
Done
UINT#1
NodeAdr
Busy
FALSE
IsEnable
Error
ErrorID
Exclusive control of instructions
EC_ChangeEnableSetting_DevC.Busy
ExclusiveFlg
S
EC_ChangeEnableSetting_DevC.Busy
ExclusiveFlg
R
SlaveDisableDone_DevC
Operating1End
DoneHold_DevC
DoneHold_DevC
Change axis to unused axis and disable EtherCAT slave D.
MC_ChangeAxisUse_DevD
MC_ChangeAxisUse
Operating1
DoneHold_DevC
DoneHold_DevD
MC_Axis001
Axis
Axis
AxisUnuseDone_DevD
Execute
Done
_mcUnusedAxis
AxisUse
Busy
CommandAborted
Error
ErrorID
2
EC_ChangeEnableSetting_DevD
2
AxisUnuseDone_DevD
ExclusiveFlg
EC_ChangeEnableSetting
SlaveDisableDone_DevD
Execute
Done
UINT#2
NodeAdr
Busy
FALSE
IsEnable
Error
ErrorID


Exclusive control of instructions
EC_ChangeEnableSetting_DevD.Busy
ExclusiveFlg
S
EC_ChangeEnableSetting_DevD.Busy
ExclusiveFlg
R
SlaveDisableDone_DevD
Operating1End
DoneHold_DevD
DoneHold_DevD
Change axis to unused axis
and disable EtherCAT slave E.
MC_ChangeAxisUse_DevE
MC_ChangeAxisUse
Operating1
DoneHold_DevD
DoneHold_DevE
MC_Axis002
Axis
Axis
AxisUnuseDone_DevE
Execute
Done
_mcUnusedAxis
AxisUse
Busy
CommandAborted
Error
ErrorID
3
EC_ChangeEnableSetting_DevE
EC_ChangeEnableSetting
3
AxisUnuseDone_DevE
ExclusiveFlg
SlaveDisableDone_DevE
Execute
Done
UINT#3
NodeAdr
Busy
FALSE
IsEnable
Error
ErrorID
Exclusive control of instructions
EC_ChangeEnableSetting_DevE.Busy
ExclusiveFlg
S
EC_ChangeEnableSetting_DevE.Busy
ExclusiveFlg
R
SlaveDisableDone_DevE
Operating1End
DoneHold_DevE
DoneHold_DevE
Confirm changing axis to unused
axis and disabling EtherCAT slave E.
Operating1
DoneHold_DevE
Operating1End
Lighting removal OK lamp
Operating1End
Light1On
S

ST

Internal Vari-ables	Variable	Data type	Initial value	Comment
	Operating1End	BOOL	FALSE	Processing completed
	Trigger1	BOOL	FALSE	Execution condition
	Operating1	BOOL	FALSE	Processing
	Operating1Set	BOOL	FALSE	Processing started
	Light1On	BOOL	FALSE	Lighting removal OK lamp
	DoneHold_DevC	BOOL	FALSE	Holding completion of processing for EtherCAT slave C
	DoneHold_DevD	BOOL	FALSE	Holding completion of processing for EtherCAT slave D
	DoneHold_DevE	BOOL	FALSE	Holding completion of processing for EtherCAT slave E
	ExclusiveFlgSet	BOOL	FALSE	Instruction Exclusive Flag ON
	ExclusiveFlgReset	BOOL	FALSE	Instruction Exclusive Flag OFF
	R_TRIG_instance1	R_TRIG		
	RS_instance1	RS		
	SR_instance1	SR		
	MC_ChangeAxisUse_DevC	MC_ChangeAxisUse		
	EC_ChangeEnableSetting_DevC	EC_ChangeEnableSetting		
	R_TRIG_DevC	R_TRIG		
	F_TRIG_DevC	F_TRIG		
	RS_ExFlg_DevC	RS		
	RS_DevC	RS		
	MC_ChangeAxisUse_DevD	MC_ChangeAxisUse		
	EC_ChangeEnableSetting_DevD	EC_ChangeEnableSetting		
	R_TRIG_DevD	R_TRIG		
	F_TRIG_DevD	F_TRIG		
	RS_ExFlg_DevD	RS		
	RS_DevD	RS		
	MC_ChangeAxisUse_DevE	MC_ChangeAxisUse		
	EC_ChangeEnableSetting_DevE	EC_ChangeEnableSetting		
	R_TRIG_DevE	R_TRIG		
	F_TRIG_DevE	F_TRIG		
	RS_ExFlg_DevE	RS		
	RS_DevE	RS		

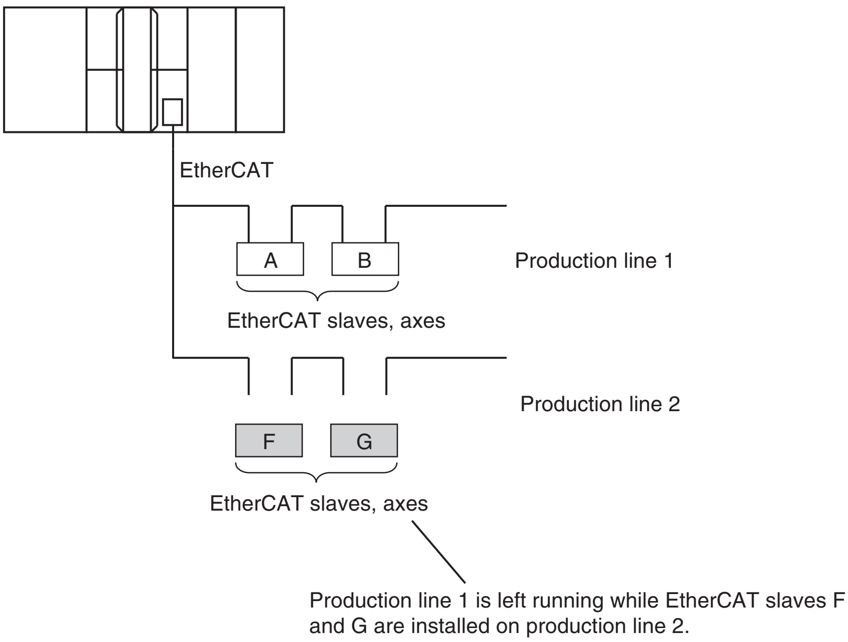
External Variables	Variable	Data type	Constant	Comment
	MC_Axis000	_sAXIS_REF	✔	Axis variable for EtherCAT slave C
	MC_Axis001	_sAXIS_REF	✔	Axis variable for EtherCAT slave D
	MC_Axis002	_sAXIS_REF	✔	Axis variable for EtherCAT slave E
	ExclusiveFlg	BOOL	---	Instruction Exclusive Flag

```
// Accept execution condition trigger.
R_TRIG_instance1(Trigger1, Operating1Set);
RS_instance1(
   Set   :=Operating1Set,
   Reset1:=Operating1End,
   Q1    =>Operating1);

// Change axis to unused axis for EtherCAT slave C.
MC_ChangeAxisUse_DevC(
   Axis   :=MC_Axis000,
   Execute:=(Operating1 & NOT(DoneHold_DevC)),
   AxisUse:=_mcUnusedAxis);

// Disable EtherCAT slave C.
EC_ChangeEnableSetting_DevC(
   Execute :=(Operating1 & MC_ChangeAxisUse_DevC.Done & NOT(ExclusiveFlg)),
   NodeAdr :=UINT#1,
   IsEnable:=FALSE);

// Exclusive control of instructions
R_TRIG_DevC(EC_ChangeEnableSetting_DevC.Busy, ExclusiveFlgSet);
F_TRIG_DevC(EC_ChangeEnableSetting_DevC.Busy, ExclusiveFlgReset);
RS_ExFlg_DevC(
   Set   :=ExclusiveFlgSet,
   Reset1:=ExclusiveFlgReset,
   Q1    =>ExclusiveFlg);
RS_DevC(
   Set   :=EC_ChangeEnableSetting_DevC.Done,
   Reset1:=Operating1End,
   Q1    =>DoneHold_DevC);

// Change axis to unused axis for EtherCAT slave D.
MC_ChangeAxisUse_DevD(
   Axis   :=MC_Axis001,
   Execute:=(Operating1 & DoneHold_DevC & NOT(DoneHold_DevD)),
   AxisUse:=_mcUnusedAxis);

// Disable EtherCAT slave D.
EC_ChangeEnableSetting_DevD(
   Execute :=(Operating1 & DoneHold_DevC & MC_ChangeAxisUse_DevD.Done &
            NOT(ExclusiveFlg)),
   NodeAdr :=UINT#2,
   IsEnable:=FALSE);

// Exclusive control of instructions
R_TRIG_DevD(EC_ChangeEnableSetting_DevD.Busy, ExclusiveFlgSet);
F_TRIG_DevD(EC_ChangeEnableSetting_DevD.Busy, ExclusiveFlgReset);
RS_ExFlg_DevD(
   Set   :=ExclusiveFlgSet,
   Reset1:=ExclusiveFlgReset,
   Q1    =>ExclusiveFlg);
RS_DevD(
   Set   :=EC_ChangeEnableSetting_DevD.Done,
   Reset1:=Operating1End,
   Q1    =>DoneHold_DevD);

// Change axis to unused axis for EtherCAT slave E.
MC_ChangeAxisUse_DevE(
   Axis   :=MC_Axis002,
   Execute:=(Operating1 & DoneHold_DevD & NOT(DoneHold_DevE)),
   AxisUse:=_mcUnusedAxis);

// Disable EtherCAT slave E.
EC_ChangeEnableSetting_DevE(
```

```
   Execute :=(Operating1 & DoneHold_DevD & MC_ChangeAxisUse_DevE.Done &
            NOT(ExclusiveFlg)),
   NodeAdr :=UINT#3,
   IsEnable:=FALSE);

// Exclusive control of instructions
R_TRIG_DevE(EC_ChangeEnableSetting_DevE.Busy, ExclusiveFlgSet);
F_TRIG_DevE(EC_ChangeEnableSetting_DevE.Busy, ExclusiveFlgReset);
RS_ExFlg_DevE(
   Set   :=ExclusiveFlgSet,
   Reset1:=ExclusiveFlgReset,
   Q1    =>ExclusiveFlg);
RS_DevE(
   Set   :=EC_ChangeEnableSetting_DevE.Done,
   Reset1:=Operating1End,
   Q1    =>DoneHold_DevE);

// Confirm changing axis to unused axis and disabling EtherCAT slave E.
Operating1End:=(Operating1 & DoneHold_DevE);

// Lighting removal OK lamp
SR_instance1(
   Set1:=Operating1End,
   Q1  =>Light1On);
```

- **Example of Connecting EtherCAT Slaves to an EtherCAT Network**

Production line 1 from the previous example is left running while EtherCAT slaves F and G are installed on production line 2. Motion control axes are set for EtherCAT slaves F and G. Therefore, the EtherCAT slaves are enabled and the axes are changed to used axes.



Procedure

The operating procedure for the sample programming is as follows:

1. The operator uses the following procedure to install EtherCAT slaves F and G.
2. The operator presses a button on an HMI to turn ON the execution condition.

3 The Controller enables EtherCAT slaves F and G. Also, the axes for those slaves are changed to used axes.

4 When enabling and changing the axes to used axes is completed for the two EtherCAT slaves, the Controller lights an installation completed lamp.

Instruction to Change Axes to Used Axes

The MC_ChangeAxisUse instruction is used to change axes to used axes. Refer to the *NJ/NX-series Motion Control Instructions Reference Manual* (Cat. No. W508) or *NY-series Motion Control Instructions Reference Manual* (Cat. No. W561) for the detailed specifications of the MC_ChangeAxisUse instruction.

Exclusive Control of Instructions

You can execute only one EC_ChangeEnableSetting instruction at the same time. Also, the EC_ChangeEnableSetting instruction is executed over more than one task. Confirm the completion of the EC_ChangeEnableSetting instruction before you execute the next EC_ChangeEnableSetting instruction. The *ExclusiveFlg* variable (Instruction Exclusive Flag) is used for this purpose. If the value of *ExclusiveFlg* is TRUE, then an EC_ChangeEnableSetting instruction is in execution. Do not execute the next EC_ChangeEnableSetting instruction while the value of *ExclusiveFlg* is TRUE.

You cannot execute the EC_ChangeEnableSetting instruction at the same time as another EC_ChangeEnableSetting instruction is in execution in another task. Therefore, *ExclusiveFlg* is defined as a global variable in this sample programming. That allows this sample programming to perform exclusive control with EC_ChangeEnableSetting instructions in the other tasks. The same global variable, *ExclusiveFlg*, must also be used in the other tasks to perform exclusive control of the instructions.

You cannot execute the EC_ChangeEnableSetting instruction at the same time as the EC_DisconnectSlave or EC_ConnectSlave instruction. The sample programming that is provided for the EC_DisconnectSlave instruction on page 2-928 uses the same *ExclusiveFlg* global variable as in this sample programming as an example of exclusive control of instructions.

Axis Variables and Node Addresses for the EtherCAT Slaves

The axis variables that are assigned to the axes for EtherCAT slaves F and G and the node addresses of the slaves are given in the following table.

EtherCAT slaves	Axis variable	Node address
F	MC_Axis003	4
G	MC_Axis004	5

Definitions of Global Variables

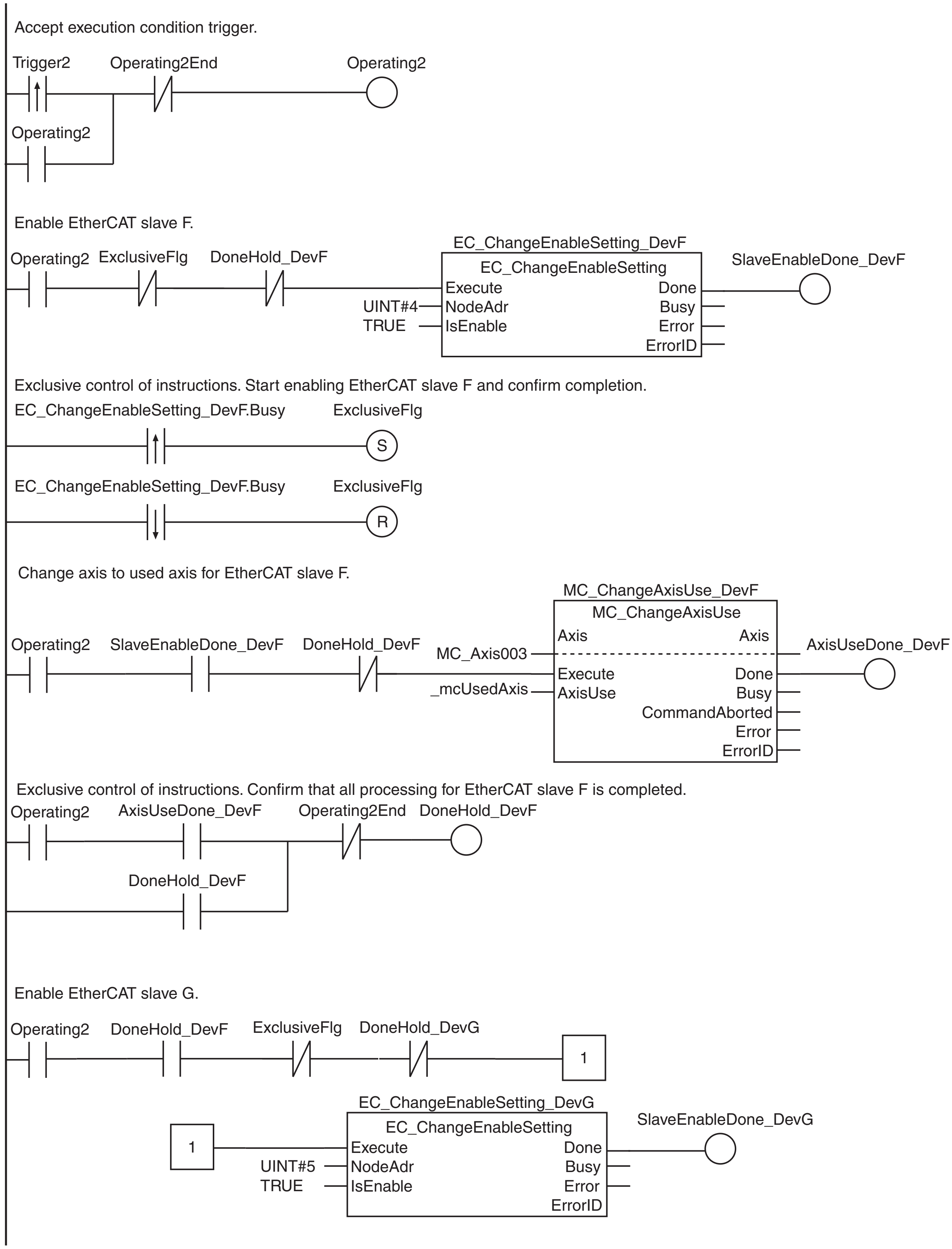
Global Variables

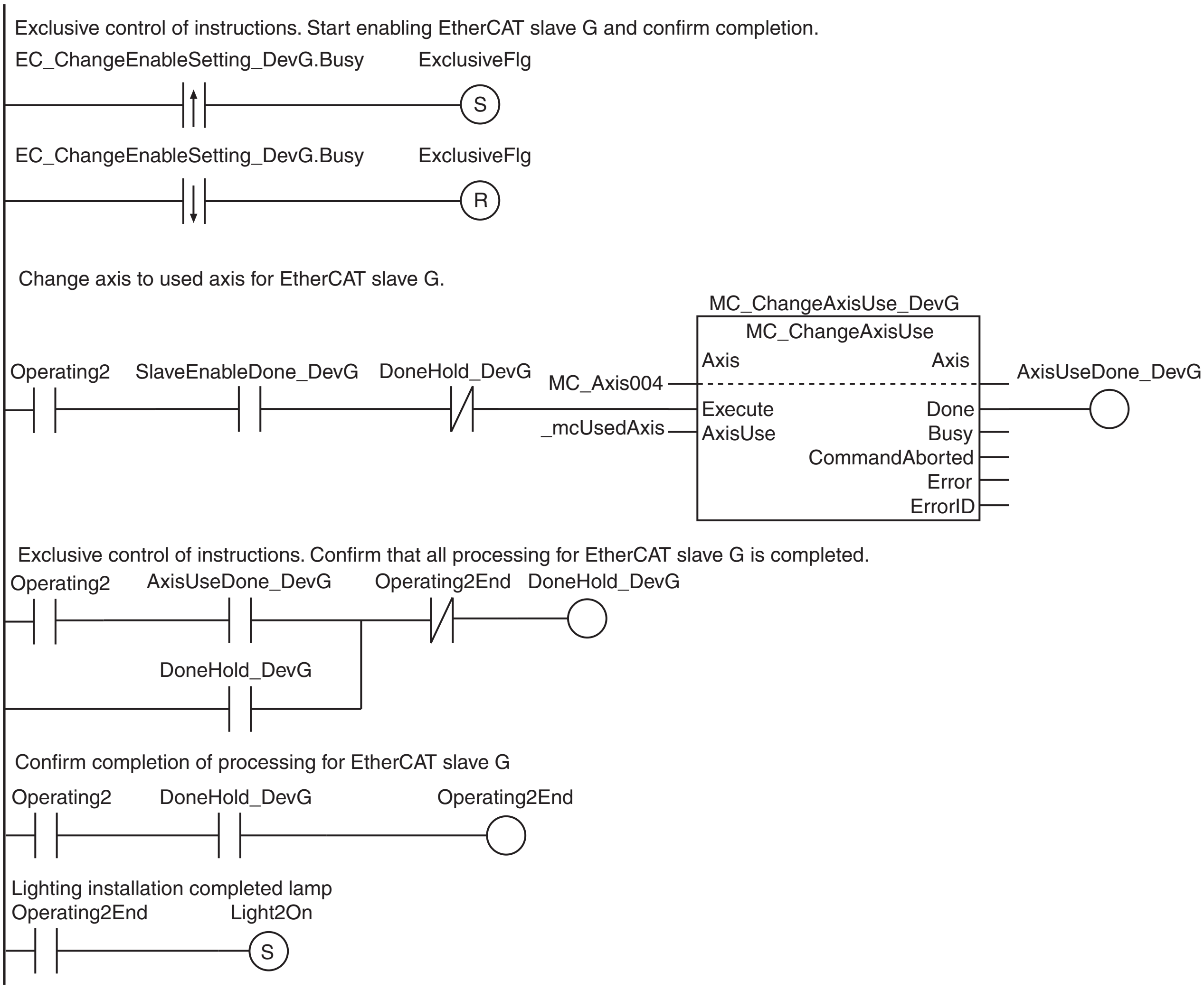
Variable	Data type	Initial value	AT specification	Constant	Comment
MC_Axis003	_sAXIS_REF		MC://_MC_AX[3]	✔	Axis variable for EtherCAT slave F
MC_Axis004	_sAXIS_REF		MC://_MC_AX[4]	✔	Axis variable for EtherCAT slave G
ExclusiveFlg	BOOL	FALSE		---	Instruction Exclusive Flag

LD

Internal Variables	Variable	Data type	Initial value	Comment
	Operating2End	BOOL	FALSE	Processing completed
	Trigger2	BOOL	FALSE	Execution condition
	Operating2	BOOL	FALSE	Processing
	AxisUseDone_DevF	BOOL	FALSE	Changing axis to used axis completed for EtherCAT slave F
	SlaveEnableDone_DevF	BOOL	FALSE	Enabling EtherCAT slave F completed
	DoneHold_DevF	BOOL	FALSE	Holding completion of processing for EtherCAT slave F
	AxisUseDone_DevG	BOOL	FALSE	Changing axis to used axis completed for EtherCAT slave G
	SlaveEnableDone_DevG	BOOL	FALSE	Enabling EtherCAT slave G completed
	DoneHold_DevG	BOOL	FALSE	Holding completion of processing for EtherCAT slave G
	Light2On	BOOL	FALSE	Lighting installation completed lamp
	MC_ChangeAxisUse_DevF	MC_ChangeAxisUse		
	EC_ChangeEnableSetting_DevF	EC_ChangeEnableSetting		
	MC_ChangeAxisUse_DevG	MC_ChangeAxisUse		
	EC_ChangeEnableSetting_DevG	EC_ChangeEnableSetting		

External Variables	Variable	Data type	Constant	Comment
	MC_Axis003	_sAXIS_REF	✔	Axis variable for EtherCAT slave F
	MC_Axis004	_sAXIS_REF	✔	Axis variable for EtherCAT slave G
	ExclusiveFlg	BOOL	---	Instruction Exclusive Flag


Accept execution condition trigger.
Trigger2
Operating2End
Operating2
Operating2
Enable EtherCAT slave F.
Operating2
ExclusiveFlg
DoneHold_DevF
EC_ChangeEnableSetting_DevF
EC_ChangeEnableSetting
Execute
NodeAdr
IsEnable
Done
Busy
Error
ErrorID
UINT#4
TRUE
SlaveEnableDone_DevF
Exclusive control of instructions. Start enabling EtherCAT slave F and confirm completion.
EC_ChangeEnableSetting_DevF.Busy
ExclusiveFlg
S
EC_ChangeEnableSetting_DevF.Busy
ExclusiveFlg
R
Change axis to used axis for EtherCAT slave F.
MC_ChangeAxisUse_DevF
MC_ChangeAxisUse
Operating2
SlaveEnableDone_DevF
DoneHold_DevF
MC_Axis003
_mcUsedAxis
Axis
Axis
Execute
AxisUse
Done
Busy
CommandAborted
Error
ErrorID
AxisUseDone_DevF
Exclusive control of instructions. Confirm that all processing for EtherCAT slave F is completed.
Operating2
AxisUseDone_DevF
Operating2End
DoneHold_DevF
DoneHold_DevF
Enable EtherCAT slave G.
Operating2
DoneHold_DevF
ExclusiveFlg
DoneHold_DevG
1
1
EC_ChangeEnableSetting_DevG
EC_ChangeEnableSetting
Execute
NodeAdr
IsEnable
Done
Busy
Error
ErrorID
UINT#5
TRUE
SlaveEnableDone_DevG



ST

Internal Vari-ables	Variable	Data type	Initial value	Comment
	Operating2End	BOOL	FALSE	Processing com-pleted
	Trigger2	BOOL	FALSE	Execution condition
	Operating2	BOOL	FALSE	Processing
	Operating2Set	BOOL	FALSE	Processing started
	Light2On	BOOL	FALSE	Lighting installation completed lamp
	DoneHold_DevF	BOOL	FALSE	Holding completion of processing for EtherCAT slave F
	DoneHold_DevG	BOOL	FALSE	Holding completion of processing for EtherCAT slave G
	ExclusiveFlgSet	BOOL	FALSE	Instruction Exclu-sive Flag ON
	ExclusiveFlgReset	BOOL	FALSE	Instruction Exclu-sive Flag OFF
	R_TRIG_instance2	R_TRIG		
	RS_instance2	RS		
	SR_instance2	SR		
	MC_ChangeAxisUse_DevF	MC_ChangeAxisUse		

Internal Vari-ables	Variable	Data type	Initial value	Comment
	EC_ChangeEnableSetting_DevF	EC_ChangeEnableSetting		
	R_TRIG_DevF	R_TRIG		
	F_TRIG_DevF	F_TRIG		
	RS_ExFlg_DevF	RS		
	RS_DevF	RS		
	MC_ChangeAxisUse_DevG	MC_ChangeAxisUse		
	EC_ChangeEnableSetting_DevG	EC_ChangeEnableSetting		
	R_TRIG_DevG	R_TRIG		
	F_TRIG_DevG	F_TRIG		
	RS_ExFlg_DevG	RS		
	RS_DevG	RS		

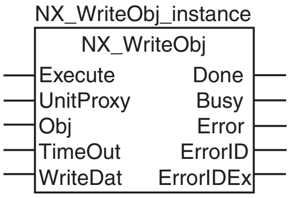
External Variables	Variable	Data type	Constant	Comment
	MC_Axis003	_sAXIS_REF	✔	Axis variable for EtherCAT slave F
	MC_Axis004	_sAXIS_REF	✔	Axis variable for EtherCAT slave G
	ExclusiveFlg	BOOL	---	Instruction Exclusive Flag

```
// Accept execution condition trigger.
R_TRIG_instance2(Trigger2, Operating2Set);
RS_instance2(
   Set   :=Operating2Set,
   Reset1:=Operating2End,
   Q1    =>Operating2);

// Enable EtherCAT slave F.
EC_ChangeEnableSetting_DevF(
   Execute :=(Operating2 & NOT(ExclusiveFlg) & NOT(DoneHold_DevF)),
   NodeAdr :=UINT#4,
   IsEnable:=TRUE);

// Exclusive control of instructions. Start enabling EtherCAT slave F and confirm
// completion.
R_TRIG_DevF(EC_ChangeEnableSetting_DevF.Busy, ExclusiveFlgSet);
F_TRIG_DevF(EC_ChangeEnableSetting_DevF.Busy, ExclusiveFlgReset);
RS_ExFlg_DevF(
   Set   :=ExclusiveFlgSet,
   Reset1:=ExclusiveFlgReset,
   Q1    =>ExclusiveFlg);

// Change axis to used axis for EtherCAT slave F.
MC_ChangeAxisUse_DevF(
   Axis   :=MC_Axis003,
   Execute:=(Operating2 & EC_ChangeEnableSetting_DevF.Done & NOT(DoneHold_DevF)),
   AxisUse:=_mcUsedAxis);

// Exclusive control of instructions. Confirm that all processing for EtherCAT
// slave F is completed.
RS_DevF(
   Set   :=(Operating2 & MC_ChangeAxisUse_DevF.Done),
   Reset1:=Operating2End,
   Q1    =>DoneHold_DevF);

// Enable EtherCAT slave G.
EC_ChangeEnableSetting_DevG(
```

```
   Execute :=(Operating2 & DoneHold_DevF & NOT(ExclusiveFlg) &
            NOT(DoneHold_DevG)),
   NodeAdr :=UINT#5,
   IsEnable:=TRUE);

// Exclusive control of instructions. Start enabling EtherCAT slave F and confirm
// completion.
R_TRIG_DevG(EC_ChangeEnableSetting_DevG.Busy, ExclusiveFlgSet);
F_TRIG_DevG(EC_ChangeEnableSetting_DevG.Busy, ExclusiveFlgReset);
RS_ExFlg_DevG(
   Set   :=ExclusiveFlgSet,
   Reset1:=ExclusiveFlgReset,
   Q1    =>ExclusiveFlg);

// Change axis to used axis for EtherCAT slave G.
MC_ChangeAxisUse_DevG(
   Axis   :=MC_Axis004,
   Execute:=(Operating2 & EC_ChangeEnableSetting_DevG.Done & NOT(DoneHold_DevG)),
   AxisUse:=_mcUsedAxis);

// Exclusive control of instructions. Confirm that all processing for EtherCAT
// slave G is completed.
RS_DevG(
   Set   :=(Operating2 & MC_ChangeAxisUse_DevG.Done),
   Reset1:=Operating2End,
   Q1    =>DoneHold_DevG);

// Confirm completion of processing for EtherCAT slave G
Operating2End:=Operating2 & DoneHold_DevG;

// Lighting installation completed lamp
SR_instance2(
   Set1:=Operating2End,
   Q1  =>Light2On);
```

NX_WriteObj

The NX_WriteObj instruction writes data to an NX object in an EtherCAT Coupler Unit or NX Unit.

Instruction	Name	FB/FUN	Graphic expression	ST expression
NX_WriteObj	Write NX Unit Object	FB		NX_WriteObj_instance(Execute, UnitProxy, Obj, TimeOut, WriteDat, Done, Busy, Error, ErrorID, ErrorIDEx);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
UnitProxy	Specified Unit	Input	Unit to which to write data	---	---	*
Obj	Object parameter		Object parameter			---
TimeOut	Timeout time		Timeout time If 0 is set, the timeout time is 2.0 s.	0 to 60,000	ms	2000 (2.0s)
WriteDat	Write data		Data to write to NX object	Depends on data type.	---	*

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
UnitProxy	Refer to *Function* for details on the structure _sNXUNIT_ID.																			
Obj	Refer to *Function* for details on the structure _sNXOBJ_ACCESS.																			
TimeOut							OK													
WriteDat	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	An array can also be specified.																			

Function

The NX_WriteObj instruction writes the contents of *WriteDat* to an NX object in an EtherCAT Coupler Unit, an NX Unit on the EtherCAT Coupler Unit, or an NX Unit connected to the NX bus of the CPU Unit. The Unit for which to write the data is specified with *UnitProxy.*

TimeOut specifies the timeout time. If a response does not return within the timeout time, it is assumed that communications failed. In that case, the data is not written.

The data type of *UnitProxy* is structure _sNXUNIT_ID. The meanings of the members are as follows:

Name	Meaning	Content	Data type
UnitProxy	Specified Unit	Specified Unit	_sNXUNIT_ID
NodeAdr	Node address	Node address of the Communications Coupler Unit	UINT
IPAdr	IP address	IP address of the Communications Coupler Unit	BYTE[5]
UnitNo	Unit number	Unit number of specified Unit	UDINT
Path	Path	Path information to the specified Unit	BYTE[64]
PathLength	Valid path length	Valid path length	USINT

Pass the device variable that is assigned to the specified Unit to *UnitProxy*.

The data type of *Obj* is structure _sNXOBJ_ACCESS. The meanings of the members are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Obj	Object parameter	Object parameter	_sNXOBJ _ACCESS	---	---	---
Index	Index	Index	UINT	Depends on data type.	---	0
Subindex	Subindex	Subindex	USINT			
IsCompleteAccess*1	Complete access	Complete access	BOOL	FALSE only		FALSE

*1 This member is not used for this instruction. Always set the value to FALSE.

Related Instructions and Execution Procedure

Depending on the attributes of the data that you write to an EtherCAT Coupler Unit, an NX Unit on the EtherCAT Coupler Unit, or an NX Unit connected to the NX bus of the CPU Unit, you must execute this instruction along with other instructions. The procedures for each case are given below.

● Execution Procedure 1

Use the following procedure to write data with the following attributes.

- Power OFF Retain attribute
- The values are updated when the Unit is restarted.

1 Use the NX_ChangeWriteMode instruction (page 2-851) to change the Unit to a mode that allows writing data.

2 Use the NX_WriteObj instruction to write data to the Unit.

3 Use the NX_SaveParam instruction (page 2-856) to save the data that you wrote.

4 Use the RestartNXUnit instruction (page 2-844) to restart the Unit.

● Execution Procedure 2

Use the following procedure to write data with the following attributes.

- Power OFF Retain attribute
- The values are updated as soon as they are written.

1 Use the NX_WriteObj instruction to write data to the Unit.

2 Use the NX_SaveParam instruction (page 2-856) to save the data that you wrote.

● Execution Procedure 3

Use the following procedure to write data with the following attributes.

- No Power OFF Retain attribute

1 Use the NX_WriteObj instruction to write data to the Unit.

Notation Example

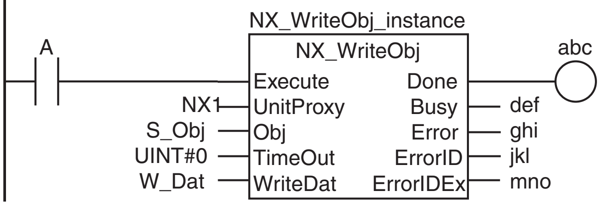
The following notation example shows how to set the NX-OD4121 Digital Output Unit to hold the present value of the output when the load becomes disconnected.

A variable that is named ‘NX1’ with a data type of _sNXUNIT_ID is assigned to the Unit to which to write the data.

For the NX-OD4121, the index of the Load OFF Output Setting parameter is UINT#16#5011 and the subindex is USINT#1.

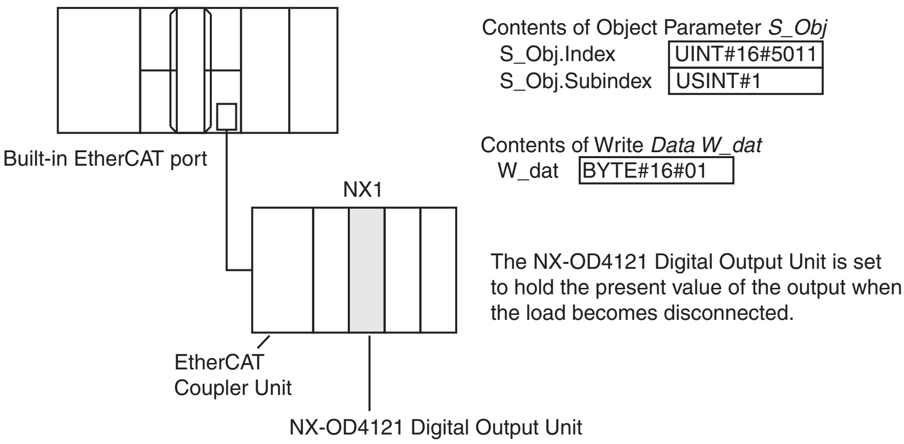
To hold the present value, BYTE#16#01 is written to the Load Rejection Output Setting parameter.

LD



ST

```
NX_WriteObj_instance(A, NX1, S_Obj, UINT#0,
                    W_Dat, abc, def, ghi, jkl, mno);
```



Related System-defined Variables

Name	Meaning	Data type	Description
_EC_MBXSlavTbl[i] "i" is the node address.	Message Communications Enabled Slave Table	BOOL	This variable indicates whether communications are possible for each slave. TRUE: Communications are possible. FALSE: Communications are not possible.
_NXB_UnitMsgActiveTbl [i]	_NXB_UnitMsgActiveTbl [i]	BOOL	This table indicates the slaves that can perform message communications. Use this variable to confirm that communications with the relevant slave are possible.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* (page *2-3*) for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- If *WriteDat* is an array, make sure that the overall size of the array is the same as the size of the NX object to write in the specified Unit.
- For *UnitProxy,* specify the device variable that is assigned to the EtherCAT Coupler Unit, an NX Unit on the EtherCAT Coupler Unit, or an NX Unit connected to the NX bus of the CPU Unit in the I/O Map of the Sysmac Studio. Refer to the *Sysmac Studio Version 1 Operation Manual* (Cat. No. W504-E1-07 or later) for details on assigning device variables.
- Always use a variable for the parameter to pass to *WriteDat*. A building error will occur if a constant is passed.
- To write and save data with a Power OFF Retain attribute, execute the NX_SaveParam instruction (page 2-856) after you execute the NX_WriteObj instruction. If you restart the Unit before you execute the NX_SaveParam instruction, the previous NX object data is restored.
- This instruction is related to NX Message Communications Errors. If too many instructions that are related to NX Message Communications Errors are executed at the same time, an NX Message Communications Error will occur. Refer to *Instructions Related to NX Message Communications Errors* (page A-20) for a list of the instructions that are related to NX Message Communications Errors.

- *Error* is TRUE if an error occurred. The meanings of the values of *ErrorID* and *ErrorIDEx* are given in the following table.

Value of *ErrorID*	Value of *ErrorIDEx*	Meaning
16#0400	16#0000 0000	• The value of *UnitProxy* is outside of the valid range. • The value of *TimeOut* is outside of the valid range.
16#0419	16#0000 0000	• The data type of *UnitProxy* is not correct. • The data type of *WriteDat* is not correct.
16#041B	16#0000 0000	More than 2,048 bytes of data was specified for *WriteDat.*
16#2C00	16#0000 0401	The specified Unit does not support the instruction.
	16#0000 1001 16#0000 1002 16#0017 0000 16#0020 0000 16#0021 0000	An input parameter, output parameter, or in-out parameter is incorrect. Confirm that the intended parameter is used for the input parameter, output parameter, or in-out parameter.
	16#0000 1010	The data size of the specified NX object does not agree with the data size specified in *WriteDat*.
	16#0000 1101	The correct Unit was not specified. Check the Unit.
	16#0000 110B	The size of the read data is too large. Make sure that the read data specification is correct.
	16#0000 1110	There is no object that corresponds to the value of *Obj.Index.*
	16#0000 1111	There is no object that corresponds to the value of *Obj.Subindex.*
	16#0000 2101	The specified NX object cannot be written.
	16#0000 2110	The value of *WriteDat* exceeds the range of the values of the NX object to write.
	16#0000 2210	The specified Unit is not in a mode that allows writing data.
	16#0000 2213	Instruction execution was not possible because the specified Unit was performing an I/O check. Execute the instruction after the I/O check is completed.
	16#0000 2230	The status of the specified Unit does not agree with the value of the read source or write destination NX object. Take the following actions if the value of *Obj.Index* is between 0x6000 and 0x6FFF or between 0x7000 and 0x7FFF. • Delete the read source or write designation NX object from the I/O allocation settings. • Reset the error for the specified Unit. • Place the specified Unit in a mode that does not allow writing data.
	16#0000 2231	Instruction execution was not possible because the specified Unit was performing initialization. Wait for the Unit to start normal operation and then execute the instruction.
	16#0000 250F	Hardware access failed. Execute the instruction again.
	16#0000 2601 16#0000 2602 16#0010 0000	The specified Unit does not support this instruction. Check the version of the Unit.
	16#0000 2603	Execution of the instruction failed. Execute the instruction again. Make sure that at least one channel is enabled in the selections of the channels to use.
	16#0000 2621	The NX Unit is not in a status in which it can acknowledge the instruction. Wait for a while and then execute the instruction again.
	16#0001 0000	The specified Unit does not exist. Make sure that the Unit configuration is correct.
	16#0011 0000	The specified port number does not exist. Make sure that the Unit configuration is correct.

Value of *ErrorID*	Value of *ErrorIDEx*	Meaning
16#2C00	16#0012 0000 16#0013 0000 16#0015 0000 16#0016 0000	The value of *UnitProxy* is not correct. Set the variable that indicates the specified EtherCAT Coupler Unit again.
	16#0014 0000	The specified node address is not correct. Make sure that the Unit configuration is correct.
	16#0030 0000 16#8001 0000	The specified Unit is busy. Execute the instruction again.
	16#0031 0000	The specified Unit not supported for connection. Check the version of the Unit.
	16#80000000 16#80050000 16#81010000 16#81020000 16#82020000 16#82030000 16#82060000 to 16#8FFF0000 16#90010000 to 16#FFFE0000	An error occurred in the communications network. Execute the instruction again.
	16#8002 0000 16#8003 0000 16#8103 0000 16#8200 0000	An error occurred in the communications network. Reduce the amount of communications traffic.
	16#8004 0000 16#8100 0000 16#8201 0000 16#8204 0000 16#8205 0000 16#9000 0000	An error occurred in the communications network. Check the Unit and cable connections. Make sure that the power supply to the Unit is ON.
16#2C01	16#0000 0000	The number of instructions that can be simultaneously executed was exceeded.
16#2C02	16#0000 0000	A timeout occurred during communications.
16#2C03	16#0000 0000	The size of the send message is not correct.

A CPU Unit with unit version 1.05 or later and Sysmac Studio version 1.06 or higher are required to use this instruction.

Sample Programming

● Example for Writing Data with Power OFF Retain Attribute That Is Updated at Unit Restart for an NX Unit

The following programming sets the Ch1 Input Moving Average Time object parameter for an NX-AD2203 AC Input Unit connected to an EtherCAT Coupler Unit to 500 μs.

The node address of the EtherCAT Coupler Unit is 10.

The specifications of the Ch1 Input Moving Average Time object parameter are as follows:

Item	Value
Index	16#5004
Subindex	16#01
Setting for 500 μs	2

The Ch1 Input Moving Average Time object parameter has a Power OFF Retain Attribute and it is updated when the Unit is restarted. Therefore, the following procedure is used.

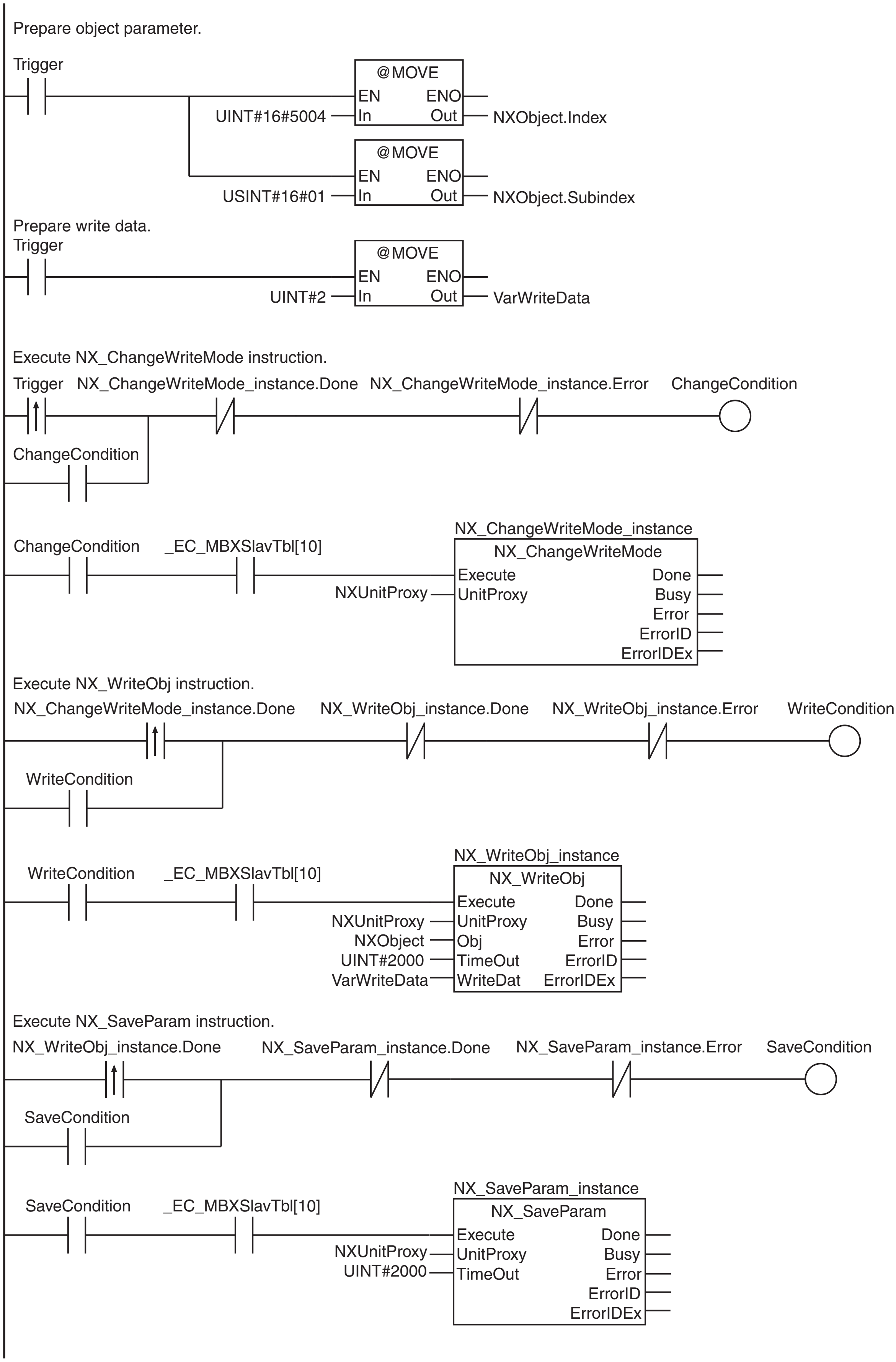
1. Use the NX_ChangeWriteMode instruction to change the Unit to a mode that allows writing data.
2. Use the NX_WriteObj instruction to write data to the Unit.
3. Use the NX_SaveParam instruction to save the data that you wrote.
4. Use the RestartNXUnit instruction to restart the Unit.

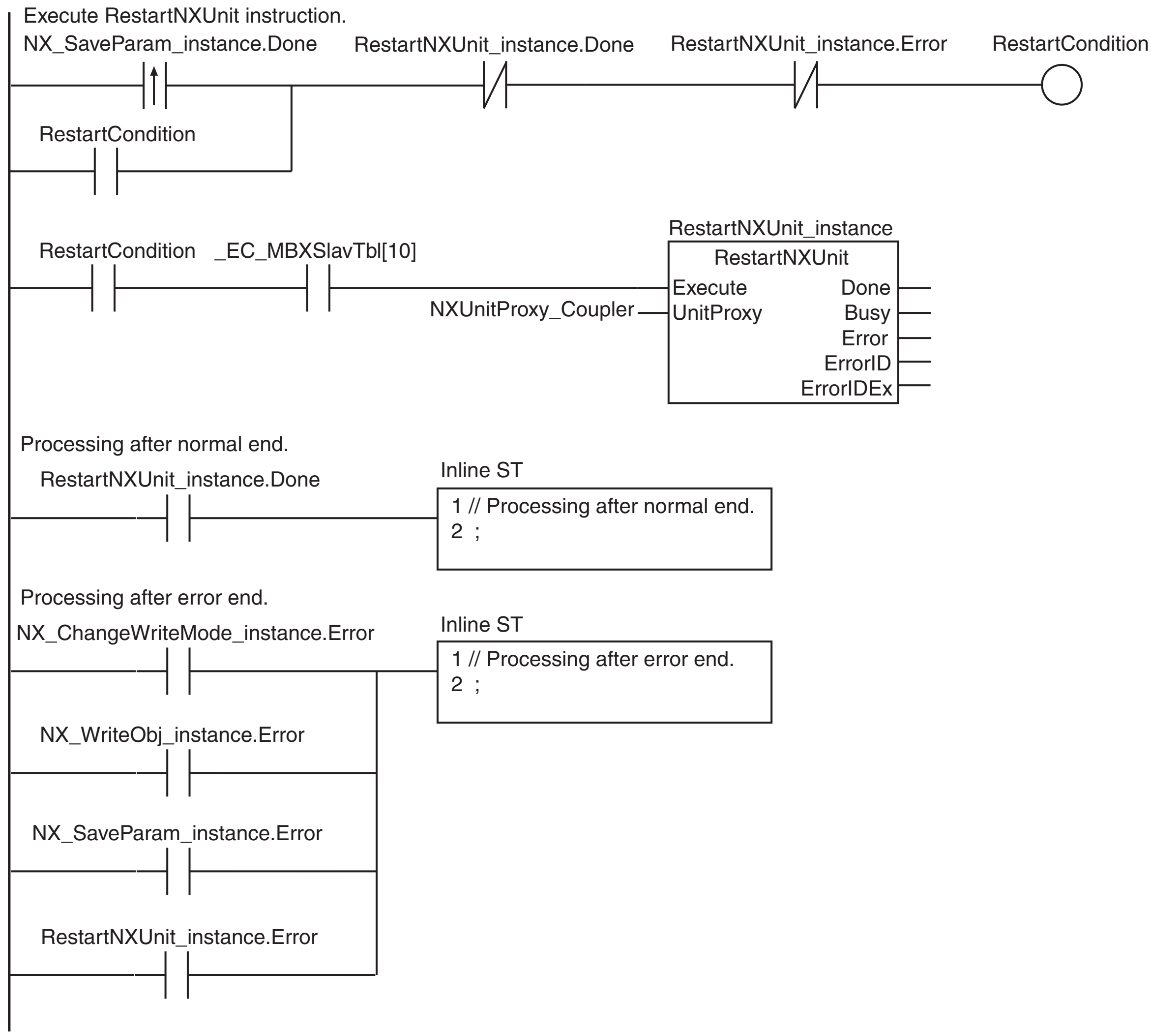
LD

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	FALSE	Execution condition
	ChangeCondition	BOOL	FALSE	Execution condition to change write mode
	WriteCondition	BOOL	FALSE	Execution condition to write data
	SaveCondition	BOOL	FALSE	Execution condition to save data
	RestartCondition	BOOL	FALSE	Execution condition to restart Unit
	NXUnitProxy	_sNXUNIT_ID		Unit designation for DC Input Unit
	NXUnitProxy_Coupler	_sNXUNIT_ID		Unit designation for EtherCAT Coupler Unit
	NXObject	_sNXOBJ_ACCESS	(Index:=0, Subindex:=0, IsCompleteAccess:=FALSE)	Object parameter
	VarWriteData	UINT	0	Write data
	NX_ChangeWriteMode_instance	NX_ChangeWriteMode		
	NX_WriteObj_instance	NX_WriteObj		
	NX_SaveParam_instance	NX_SaveParam		
	RestartNXUnit_instance	RestartNXUnit		

External Variables	Variable	Constant	Data type	Comment
	_EC_MBXSlavTbl	✔	ARRAY[1..512] OF BOOL *1	Message Communications Enabled Slave Table

*1 The data type is ARRAY [1..192] OF BOOL for an NJ-series CPU Unit.


Prepare object parameter.
Trigger
@MOVE
EN
ENO
In
Out
UINT#16#5004
NXObject.Index
@MOVE
EN
ENO
In
Out
USINT#16#01
NXObject.Subindex
Prepare write data.
Trigger
@MOVE
EN
ENO
In
Out
UINT#2
VarWriteData
Execute NX_ChangeWriteMode instruction.
Trigger
NX_ChangeWriteMode_instance.Done
NX_ChangeWriteMode_instance.Error
ChangeCondition
ChangeCondition
ChangeCondition
_EC_MBXSlavTbl[10]
NX_ChangeWriteMode_instance
NX_ChangeWriteMode
Execute
UnitProxy
Done
Busy
Error
ErrorID
ErrorIDEx
NXUnitProxy
Execute NX_WriteObj instruction.
NX_ChangeWriteMode_instance.Done
NX_WriteObj_instance.Done
NX_WriteObj_instance.Error
WriteCondition
WriteCondition
WriteCondition
_EC_MBXSlavTbl[10]
NX_WriteObj_instance
NX_WriteObj
Execute
UnitProxy
Obj
TimeOut
WriteDat
Done
Busy
Error
ErrorID
ErrorIDEx
NXUnitProxy
NXObject
UINT#2000
VarWriteData
Execute NX_SaveParam instruction.
NX_WriteObj_instance.Done
NX_SaveParam_instance.Done
NX_SaveParam_instance.Error
SaveCondition
SaveCondition
SaveCondition
_EC_MBXSlavTbl[10]
NX_SaveParam_instance
NX_SaveParam
Execute
UnitProxy
TimeOut
Done
Busy
Error
ErrorID
ErrorIDEx
NXUnitProxy
UINT#2000


Execute RestartNXUnit instruction.
NX_SaveParam_instance.Done
RestartNXUnit_instance.Done
RestartNXUnit_instance.Error
RestartCondition
RestartCondition
RestartCondition
_EC_MBXSlavTbl[10]
RestartNXUnit_instance
RestartNXUnit
Execute
UnitProxy
Done
Busy
Error
ErrorID
ErrorIDEx
NXUnitProxy_Coupler
Processing after normal end.
RestartNXUnit_instance.Done
Inline ST
1 // Processing after normal end.
2 ;
Processing after error end.
NX_ChangeWriteMode_instance.Error
Inline ST
1 // Processing after error end.
2 ;
NX_WriteObj_instance.Error
NX_SaveParam_instance.Error
RestartNXUnit_instance.Error

ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	FALSE	Execution condition
	ChangeCondition	BOOL	FALSE	Execution condition to change write mode
	ChangeGo	BOOL	FALSE	Execution of change to write mode
	WriteCondition	BOOL	FALSE	Execution condition to write data
	WriteGo	BOOL	FALSE	Execution of data write
	SaveCondition	BOOL	FALSE	Execution condition to save data
	SaveGo	BOOL	FALSE	Execution of data save
	RestartCondition	BOOL	FALSE	Execution condition to restart Unit
	RestartGo	BOOL	FALSE	Execution of Unit restart
	NXUnitProxy	_sNXUNIT_ID		Unit designation for DC Input Unit
	NXUnitProxy_Coupler	_sNXUNIT_ID		Unit designation for EtherCAT Coupler Unit
	NXObject	_sNXOBJ_ACCESS	(Index:=0, Subindex:=0, IsCompleteAccess:=FALSE)	Object parameter
	VarWriteData	UINT	0	Write data
	NormalEnd	UINT	0	Normal end
	ErrorEnd	UINT	0	Error end
	NX_ChangeWriteMode_instance	NX_ChangeWriteMode		
	NX_WriteObj_instance	NX_WriteObj		
	NX_SaveParam_instance	NX_SaveParam		
	RestartNXUnit_instance	RestartNXUnit		
	R_Trig_instance	R_TRIG		

External Variables	Variable	Constant	Data type	Comment
	_EC_MBXSlavTbl	✔	ARRAY[1..512] OF BOOL *1	Message Communications Enabled Slave Table

*1 The data type is ARRAY [1..192] OF BOOL for an NJ-series CPU Unit.

```
// Prepare object parameter and write data.
R_Trig_instance(Clk := Trigger);
IF (R_Trig_instance.Q=TRUE)THEN
   NXObject.Index    := UINT#16#5004;
   NXObject.Subindex := USINT#1;
   VarWriteData      := UINT#2;
END_IF;

// Execute NX_ChangeWriteMode instruction.
IF (Trigger = TRUE) THEN
   ChangeCondition := TRUE;
END_IF;

IF ((NX_ChangeWriteMode_instance.Done=TRUE) OR
(NX_ChangeWriteMode_instance.Error=TRUE))THEN
   ChangeCondition := FALSE;
END_IF;

ChangeGo := ChangeCondition & _EC_MBXSlavTbl[10];
NX_ChangeWriteMode_instance(
   Execute   := ChangeGo,
   UnitProxy := NXUnitProxy);
```

```
// Execute NX_WriteObj instruction.
IF (NX_ChangeWriteMode_instance.Done=TRUE) THEN
   WriteCondition := TRUE;
END_IF;

IF ((NX_WriteObj_instance.Done=TRUE) OR (NX_WriteObj_instance.Error=TRUE))THEN
   WriteCondition := FALSE;
END_IF;

WriteGo := WriteCondition & _EC_MBXSlavTbl[10];
NX_WriteObj_instance(
   Execute   := WriteGo,
   UnitProxy := NXUnitProxy,
   Obj       := NXObject,
   TimeOut   := UINT#2000,
   WriteDat  := VarWriteData);

// Execute NX_SaveParam instruction.
IF (NX_WriteObj_instance.Done=TRUE) THEN
   SaveCondition := TRUE;
END_IF;

IF ((NX_SaveParam_instance.Done=TRUE) OR (NX_SaveParam_instance.Error=TRUE))THEN
   SaveCondition := FALSE;
END_IF;

SaveGo := SaveCondition & _EC_MBXSlavTbl[10];
NX_SaveParam_instance(
   Execute   := SaveGo,
   UnitProxy := NXUnitProxy,
   TimeOut   := UINT#2000);

// Execute RestartNXUnit instruction.
IF (NX_SaveParam_instance.Done=TRUE) THEN
   RestartCondition := TRUE;
END_IF;

IF ((RestartNXUnit_instance.Done=TRUE) OR (RestartNXUnit_instance.Error=TRUE))
THEN
   RestartCondition := FALSE;
END_IF;

RestartGo := RestartCondition & _EC_MBXSlavTbl[10];
RestartNXUnit_instance(
   Execute   := SaveGo,
   UnitProxy := NXUnitProxy_Coupler);

IF (RestartNXUnit_instance.Done=TRUE) THEN
    // Processing after normal end.
    NormalEnd := NormalEnd + UINT#1;
ELSIF ((NX_ChangeWriteMode_instance.Error=TRUE) OR
(NX_WriteObj_instance.Error=TRUE)
        OR (NX_SaveParam_instance.Error=TRUE) OR
(RestartNXUnit_instance.Error=TRUE)) THEN
   // Processing after error end.
   ErrorEnd := ErrorEnd + UINT#1;
END_IF;
```

● Example for Writing Data with Power OFF Retain Attribute That Is Updated after Writing the Data

The following programming sets the Ch1 Offset Value (One-point Correction) object parameter for an NX-TS2101 Temperature Input Unit connected to an EtherCAT Coupler Unit to 0.3°C.

The node address of the EtherCAT Coupler Unit is 10.

The specifications of the Ch1 Offset Value (One-point Correction) object parameter are as follows:

Item	Value
Index	16#5010
Subindex	16#01
Value to write	0.3

The Ch1 Offset Value (One-point Correction) object parameter has a Power OFF Retain Attribute and it is updated after the data is written. Therefore, the following procedure is used.

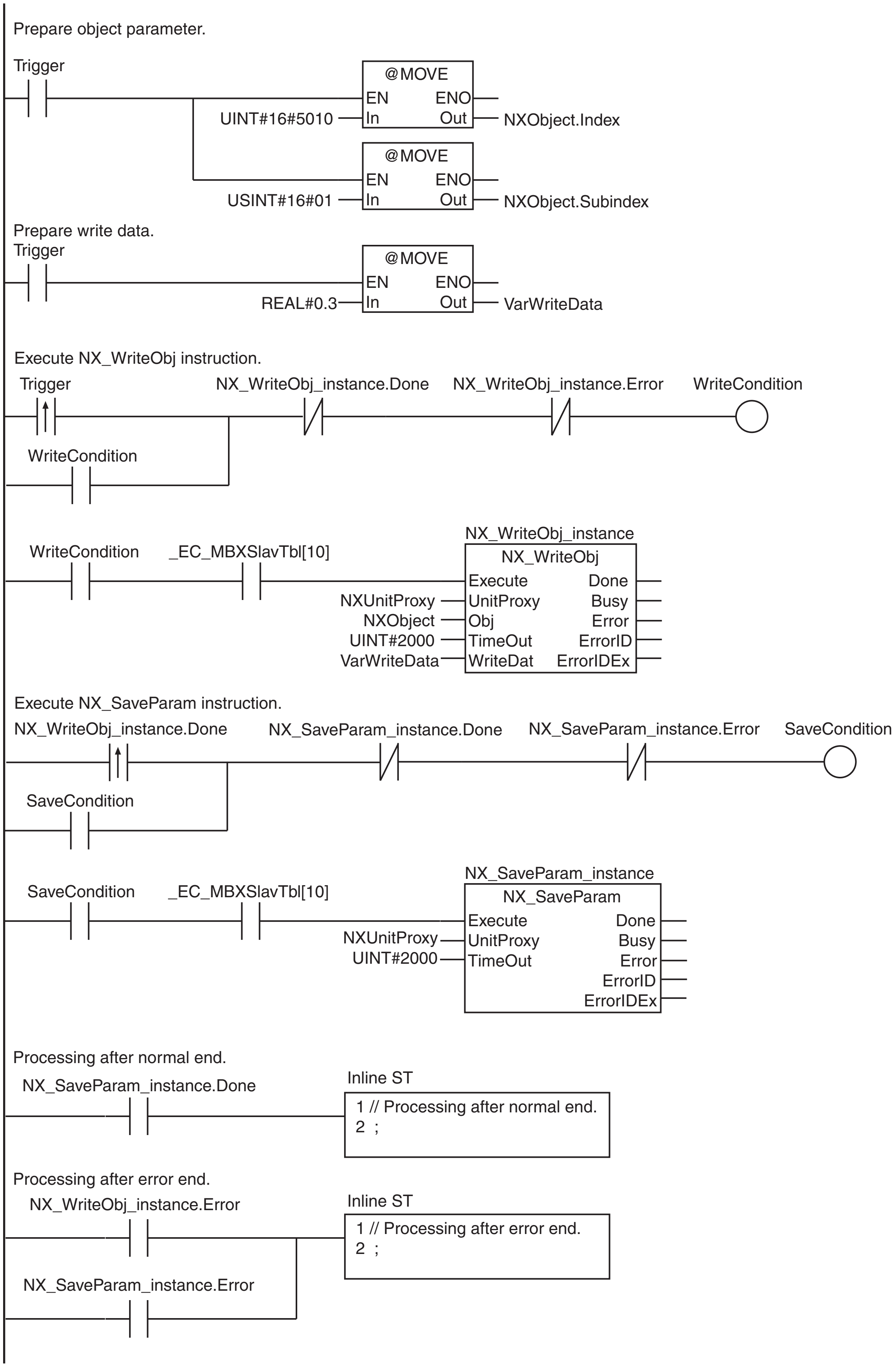
1. Use the NX_WriteObj instruction to write data to the Unit.
2. Use the NX_SaveParam instruction to save the data that you wrote.

LD

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	FALSE	Execution condition
	WriteCondition	BOOL	FALSE	Execution condition to write data
	SaveCondition	BOOL	FALSE	Execution condition to save data
	NXUnitProxy	_sNXUNIT_ID		Unit designation for AC Input Unit
	NXUnitProxy_Coupler	_sNXUNIT_ID		Unit designation for EtherCAT Coupler Unit
	NXObject	_sNXOBJ_ACCESS	(Index:=0, Subindex:=0, IsCompleteAccess:=FALSE)	Object parameter
	VarWriteData	Real	0.0	Write data
	NX_WriteObj_instance	NX_WriteObj		
	NX_SaveParam_instance	NX_SaveParam		

External Variables	Variable	Constant	Data type	Comment
	_EC_MBXSlavTbl	✔	ARRAY[1..512] OF BOOL *1	Message Communications Enabled Slave Table

*1 The data type is ARRAY [1..192] OF BOOL for an NJ-series CPU Unit.



Prepare object parameter.
Trigger
@MOVE
EN ENO
UINT#16#5010 In Out NXObject.Index
@MOVE
EN ENO
USINT#16#01 In Out NXObject.Subindex
Prepare write data.
Trigger
@MOVE
EN ENO
REAL#0.3 In Out VarWriteData
Execute NX_WriteObj instruction.
Trigger
NX_WriteObj_instance.Done
NX_WriteObj_instance.Error
WriteCondition
WriteCondition
WriteCondition
_EC_MBXSlavTbl[10]
NX_WriteObj_instance
NX_WriteObj
Execute Done
NXUnitProxy UnitProxy Busy
NXObject Obj Error
UINT#2000 TimeOut ErrorID
VarWriteData WriteDat ErrorIDEx
Execute NX_SaveParam instruction.
NX_WriteObj_instance.Done
NX_SaveParam_instance.Done
NX_SaveParam_instance.Error
SaveCondition
SaveCondition
SaveCondition
_EC_MBXSlavTbl[10]
NX_SaveParam_instance
NX_SaveParam
Execute Done
NXUnitProxy UnitProxy Busy
UINT#2000 TimeOut Error
ErrorID
ErrorIDEx
Processing after normal end.
NX_SaveParam_instance.Done
Inline ST
1 // Processing after normal end.
2 ;
Processing after error end.
NX_WriteObj_instance.Error
Inline ST
1 // Processing after error end.
2 ;
NX_SaveParam_instance.Error

ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	FALSE	Execution condition
	WriteCondition	BOOL	FALSE	Execution condition to write data
	WriteGo	BOOL	FALSE	Execution of data write
	SaveCondition	BOOL	FALSE	Execution condition to save data
	SaveGo	BOOL	FALSE	Execution of data save
	NXUnitProxy	_sNXUNIT_ID		Unit designation for Temperature Input Unit
	NXUnitProxy_Coupler	_sNXUNIT_ID		Unit designation for EtherCAT Coupler Unit
	NXObject	_sNXOBJ_ACCESS	(Index:=0, Subindex:=0, IsCompleteAccess:=FALSE)	Object parameter
	VarWriteData	REAL	0.0	Write data
	NormalEnd	UINT	0	Normal end
	ErrorEnd	UINT	0	Error end
	NX_WriteObj_instance	NX_WriteObj		
	NX_SaveParam_instance	NX_SaveParam		
	R_Trig_Instance	R_TRIG		

External Variables	Variable	Constant	Data type	Comment
	_EC_MBXSlavTbl	✔	ARRAY[1..512] OF BOOL *1	Message Communications Enabled Slave Table

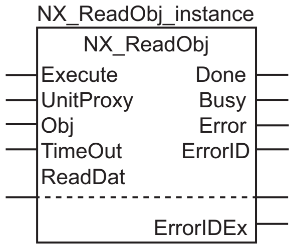
*1 The data type is ARRAY [1..192] OF BOOL for an NJ-series CPU Unit.

```
// Prepare object parameter and write data.
R_Trig_instance(Clk := Trigger);
IF (R_Trig_instance.Q=TRUE)THEN
   NXObject.Index    := UINT#16#5004;
   NXObject.Subindex := USINT#1;
   VarWriteData      := UINT#2;
END_IF;

// Execute NX_WriteObj instruction.
IF (Trigger=TRUE) THEN
   WriteCondition := TRUE;
END_IF;

IF ((NX_WriteObj_instance.Done=TRUE) OR (NX_WriteObj_instance.Error=TRUE))THEN
   WriteCondition := FALSE;
END_IF;

WriteGo := WriteCondition & _EC_MBXSlavTbl[10];
NX_WriteObj_instance(
   Execute   := WriteGo,
   UnitProxy := NXUnitProxy,
   Obj       := NXObject,
   TimeOut   := UINT#2000,
   WriteDat  := VarWriteData);

// Execute NX_SaveParam instruction.
IF (NX_WriteObj_instance.Done=TRUE) THEN
   SaveCondition := TRUE;
END_IF;

IF ((NX_SaveParam_instance.Done=TRUE) OR (NX_SaveParam_instance.Error=TRUE))THEN
```

```
   SaveCondition := FALSE;
END_IF;

SaveGo := SaveCondition & _EC_MBXSlavTbl[10];
NX_SaveParam_instance(
   Execute   := SaveGo,
   UnitProxy := NXUnitProxy,
   TimeOut   := UINT#2000);

IF (NX_SaveParam_instance.Done=TRUE) THEN
    // Processing after normal end.
    NormalEnd := NormalEnd + UINT#1;
ELSIF ((NX_WriteObj_instance.Error=TRUE) OR (NX_SaveParam_instance.Error=TRUE))
THEN
   // Processing after error end.
   ErrorEnd := ErrorEnd + UINT#1;
END_IF;
```

NX_ReadObj

The NX_ReadObj instruction reads data from an NX object in an EtherCAT Coupler Unit or NX Unit.

Instruction	Name	FB/FUN	Graphic expression	ST expression
NX_ReadObj	Read NX Unit Object	FB		NX_ReadObj_instance(Execute, UnitProxy, Obj, TimeOut, ReadDat, Done, Busy, Error, ErrorID, ErrorIDEx);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
UnitProxy	Specified Unit	Input	Unit from which to read data	---	---	*
Obj	Object parameter		Object parameter			---
TimeOut	Timeout time		Timeout time If 0 is set, the timeout time is 2.0 s.	0 to 60,000	ms	2000 (2.0 s)
ReadDat	Read data	In-out	Data read from NX object	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
UnitProxy	Refer to *Function* for details on the structure _sNXUNIT_ID.																			
Obj	Refer to *Function* for details on the structure _sNXOBJ_ACCESS.																			
TimeOut							OK													
ReadDat	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	An array can also be specified.																			

Function

The NX_ReadObj instruction reads data from an NX object in an EtherCAT Coupler Unit, an NX Unit on the EtherCAT Coupler Unit, or an NX Unit connected to the NX bus of the CPU Unit and stores the data in *ReadDat*. The Unit from which the data is read is specified with *UnitProxy.*

TimeOut specifies the timeout time. If a response does not return within the timeout time, it is assumed that communications failed. In that case, the data is not read.

The data type of *UnitProxy* is structure _sNXUNIT_ID. The meanings of the members are as follows:

Name	Meaning	Content	Data type
UnitProxy	Specified Unit	Specified Unit	_sNXUNIT_ID
NodeAdr	Node address	Node address of the Communications Coupler Unit	UINT
IPAdr	IP address	IP address of the Communications Coupler Unit	BYTE[5]
UnitNo	Unit number	Unit number of specified Unit	UDINT
Path	Path	Path information to the specified Unit	BYTE[64]
PathLength	Valid path length	Valid path length	USINT

Pass the device variable that is assigned to the specified Unit to *UnitProxy*.

The data type of *Obj* is structure _sNXOBJ_ACCESS. The meanings of the members are as follows:

Name	Meaning	Content	Data type	Valid range	Unit	Default
Obj	Object parameter	Object parameter	_sNXOBJ _ACCESS	---	---	---
Index	Index	Index	UINT	Depends on data type.	---	0
Subindex	Subindex	Subindex	USINT			
IsCompleteAccess*1	Complete access	Complete access	BOOL	FALSE only		FALSE

*1 This member is not used for this instruction. Always set the value to FALSE.

Notation Example

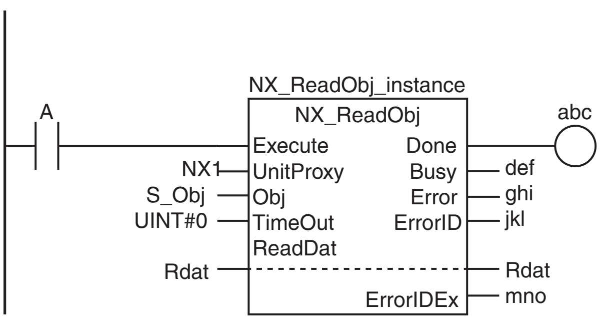
The following notation example shows how to read the unit version from an NX-ID4342 Digital Input Unit.

The read data is stored in *Rdat,* which is a UDINT variable.

A variable that is named ‘NX1’ with a data type of _sNXUNIT_ID is assigned to the Unit from which to read the data.

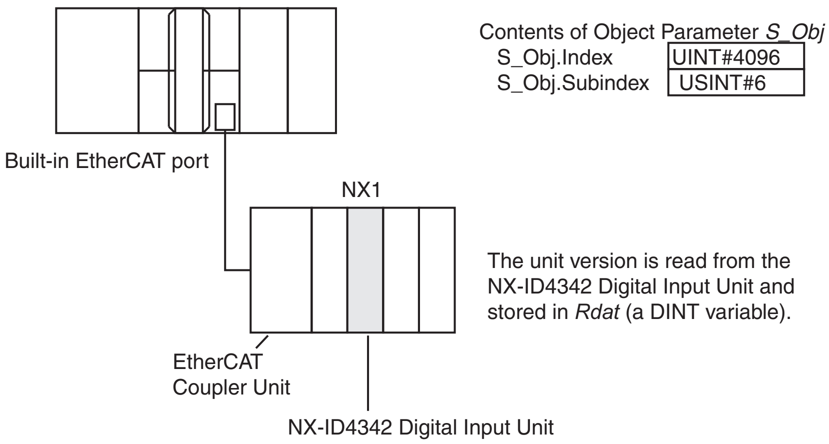
For the NX-ID4342, the index of the Unit version is UINT#16#1000 and the subindex is USINT#6.

LD



ST

```
NX_ReadObj_instance(A, NX1, S_Obj, UINT#0,
                    Rdat, abc, def, ghi, jkl, mno);
```



Related System-defined Variables

Variable	Name	Data type	Description
_EC_MBXSlavTbl[i] “i” is the node address.	Message Communications Enabled Slave Table	BOOL	This variable indicates whether communications are possible for each slave. TRUE: Communications are possible. FALSE: Communications are not possible.
_NXB_UnitMsgActiveTbl[i]	NX Unit Message Enabled Status	BOOL	This table indicates the slaves that can perform message communications. Use this variable to confirm that communications with the relevant slave are possible.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* (page 2-3) for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- If *ReadDat* is an array, make sure that the overall size of the array is the same as the size of the NX object to read in the specified Unit
- For *UnitProxy,* specify the device variable that is assigned to the EtherCAT Coupler Unit, an NX Unit on the EtherCAT Coupler Unit, or an NX Unit connected to the NX bus of the CPU Unit in the I/O Map of the Sysmac Studio. Refer to the *Sysmac Studio Version 1 Operation Manual* (Cat. No. W504-E1-07 or later) for details on assigning device variables.
- This instruction is related to NX Message Communications Errors. If too many instructions that are related to NX Message Communications Errors are executed at the same time, an NX Message Communications Error will occur. Refer to Instructions Related to NX Message Communications Errors (page A-20) for a list of the instructions that are related to NX Message Communications Errors.

- *Error* is TRUE if an error occurred. The meanings of the values of *ErrorID* and *ErrorIDEx* are given in the following table.

Value of *ErrorID*	Value of *ErrorIDEx*	Meaning
16#0400	16#0000 0000	• The value of *UnitProxy* is outside of the valid range. • The value of *TimeOut* is outside of the valid range.
16#0410	16#0000 0000	*ReadDat* is STRING data and it does not end in a NULL character.
16#0419	16#0000 0000	• The data type of *UnitProxy* is not correct. • The data type of *ReadDat* is not correct.
16#041C	16#0000 0000	The size of *ReadDat* is not the same as the size of the NX object to read.
16#2C00	16#0000 0401	The specified Unit does not support the instruction.
	16#0000 1001 16#0000 1002 16#0017 0000 16#0020 0000 16#0021 0000	An input parameter, output parameter, or in-out parameter is incorrect. Confirm that the intended parameter is used for the input parameter, output parameter, or in-out parameter.
	16#0000 1010	The data size of the specified NX object does not agree with the data size specified in *WriteDat*.
	16#0000 1101	The correct Unit was not specified. Check the Unit.
	16#0000 110B	The size of the read data is too large. Make sure that the read data specification is correct.
	16#0000 1110	There is no object that corresponds to the value of *Obj.Index.*
	16#0000 1111	There is no object that corresponds to the value of *Obj.Subindex.*
	16#0000 2101	The specified NX object cannot be written.
	16#0000 2110	The value of *WriteDat* exceeds the range of the values of the NX object to write.
	16#0000 2210	The specified Unit is not in a mode that allows writing data.
	16#0000 2213	Instruction execution was not possible because the specified Unit was performing an I/O check. Execute the instruction after the I/O check is completed.
	16#0000 2230	The status of the specified Unit does not agree with the value of the read source or write destination NX object. Take the following actions if the value of *Obj.Index* is between 0x6000 and 0x6FFF or between 0x7000 and 0x7FFF. • Delete the read source or write designation NX object from the I/O allocation settings. • Reset the error for the specified Unit. • Place the specified Unit in a mode that does not allow writing data.
	16#0000 2231	Instruction execution was not possible because the specified Unit was performing initialization. Wait for the Unit to start normal operation and then execute the instruction.
	16#0000 250F	Hardware access failed. Execute the instruction again.
	16#0000 2601 16#0000 2602 16#0010 0000	The specified Unit does not support this instruction. Check the version of the Unit.
	16#0000 2603	Execution of the instruction failed. Execute the instruction again. Make sure that at least one channel is enabled in the selections of the channels to use.
	16#0000 2621	The NX Unit is not in a status in which it can acknowledge the instruction. Wait for a while and then execute the instruction again.
	16#0001 0000	The specified Unit does not exist. Make sure that the Unit configuration is correct.

Value of *ErrorID*	Value of *ErrorIDEx*	Meaning
16#2C00	16#0011 0000	The specified port number does not exist. Make sure that the Unit configuration is correct.
	16#0012 0000 16#0013 0000 16#0015 0000 16#0016 0000	The value of *UnitProxy* is not correct. Set the variable that indicates the specified EtherCAT Coupler Unit again.
	16#0014 0000	The specified node address is not correct. Make sure that the Unit configuration is correct.
	16#0030 0000 16#8001 0000	The specified Unit is busy. Execute the instruction again.
	16#0031 0000	The specified Unit is not supported for connection. Check the version of the Unit.
	16#80000000 16#80050000 16#81010000 16#81020000 16#82020000 16#82030000 16#82060000 to 16#8FFF0000 16#90010000 to 16#FFFE0000	An error occurred in the communications network. Execute the instruction again.
	16#8002 0000 16#8003 0000 16#8103 0000 16#8200 0000	An error occurred in the communications network. Reduce the amount of communications traffic.
	16#8004 0000 16#8100 0000 16#8201 0000 16#8204 0000 16#8205 0000 16#9000 0000	An error occurred in the communications network. Check the Unit and cable connections. Make sure that the power supply to the Unit is ON.
16#2C01	16#0000 0000	The number of instructions that can be simultaneously executed was exceeded.
16#2C02	16#0000 0000	A timeout occurred during communications.

Version Information

A CPU Unit with unit version 1.05 or later and Sysmac Studio version 1.06 or higher are required to use this instruction.

Sample Programming

This sample reads the value of the I/O Refresh Method 1 object parameter from an NX-ECC201 EtherCAT Coupler Unit.

The node address of the EtherCAT Coupler Unit is 10.

The values of the index and subindex of the I/O Refresh Method 1 object parameter are as follows:

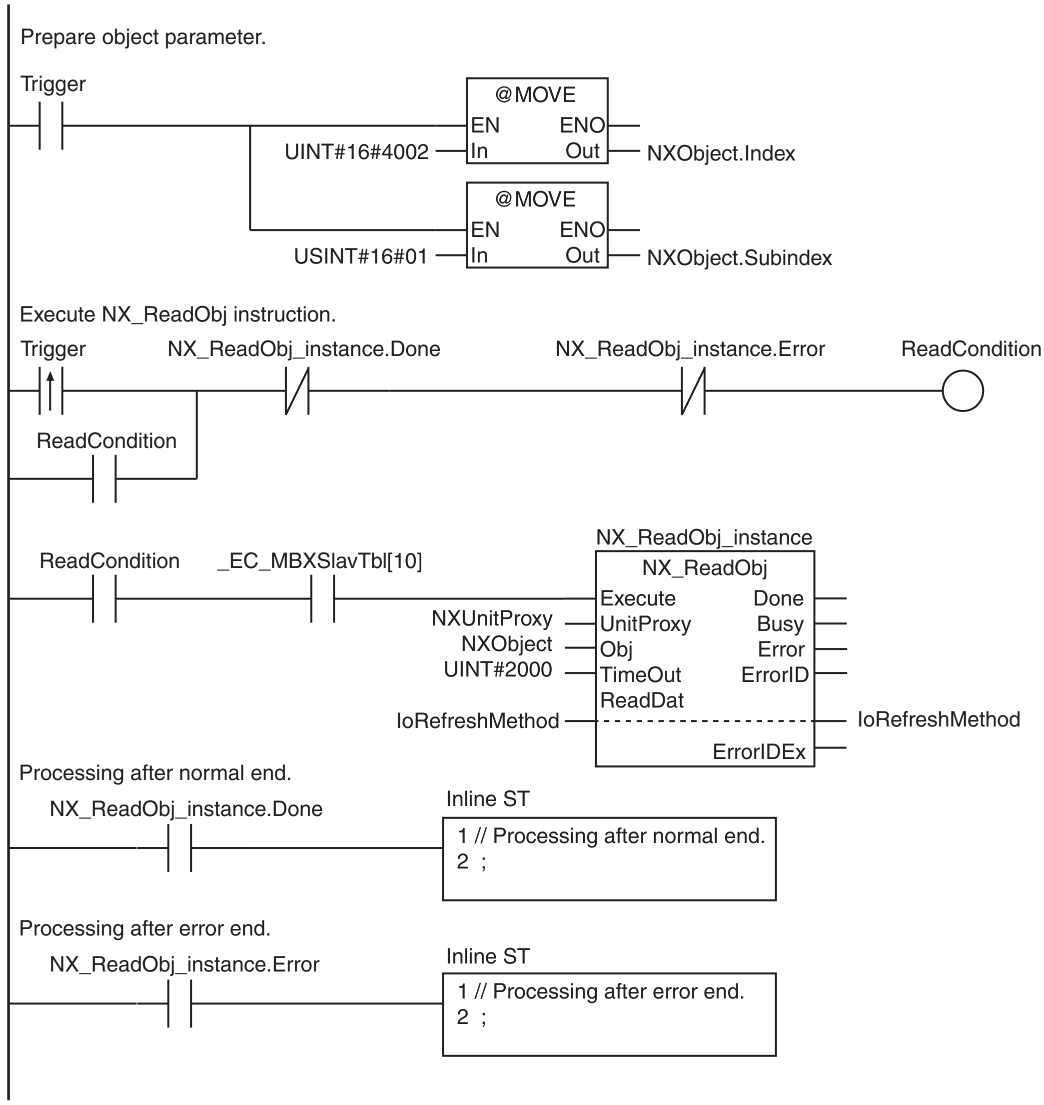
Item	Value
Index	16#4002
Subindex	16#01

LD

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	FALSE	Execution condition
	ReadCondition	BOOL	FALSE	Execution condition to read data
	NXUnitProxy	_sNXUNIT_ID		Unit designation
	NXObject	_sNXOBJ_ACCESS	(Index:=0, Subindex:=0, IsCompleteAccess:=FALSE)	Object parameter
	IoRefreshMethod	USINT	0	Read data
	NX_ReadObj_instance	NX_ReadObj		

External Variables	Variable	Constant	Data type	Comment
	_EC_MBXSlavTbl	✔	ARRAY[1..512] OF BOOL *1	Message Communications Enabled Slave Table

*1 The data type is ARRAY [1..192] OF BOOL for an NJ-series CPU Unit.


Prepare object parameter.
Trigger
@MOVE
EN ENO
In Out
UINT#16#4002
NXObject.Index
@MOVE
EN ENO
In Out
USINT#16#01
NXObject.Subindex
Execute NX_ReadObj instruction.
Trigger
NX_ReadObj_instance.Done
NX_ReadObj_instance.Error
ReadCondition
ReadCondition
ReadCondition
_EC_MBXSlavTbl[10]
NX_ReadObj_instance
NX_ReadObj
Execute Done
UnitProxy Busy
Obj Error
TimeOut ErrorID
ReadDat
ErrorIDEx
NXUnitProxy
NXObject
UINT#2000
IoRefreshMethod
IoRefreshMethod
Processing after normal end.
NX_ReadObj_instance.Done
Inline ST
1 // Processing after normal end.
2 ;
Processing after error end.
NX_ReadObj_instance.Error
Inline ST
1 // Processing after error end.
2 ;

ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	FALSE	Execution condition
	ReadCondition	BOOL	FALSE	Execution condition to read data
	ReadGo	BOOL	FALSE	Execution of data read
	NXUnitProxy	_sNXUNIT_ID		Unit designation
	NXObject	_sNXOBJ_ACCESS	(Index:=0, Subindex:=0, IsCompleteAccess:=FALSE)	Object parameter
	IoRefreshMethod	USINT	0	Read data
	NormalEnd	UINT	0	Normal end
	ErrorEnd	UINT	0	Error end
	R_Trig_instance	R_Trig		
	NX_ReadObj_instance	NX_ReadObj		

External Variables	Variable	Constant	Data type	Comment
	_EC_MBXSlavTbl	✔	ARRAY[1..512] OF BOOL *1	Message Communications Enabled Slave Table

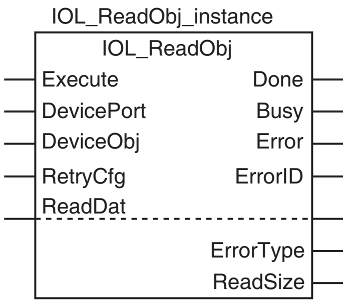
*1 The data type is ARRAY [1..192] OF BOOL for an NJ-series CPU Unit.

```
// Prepare object parameter.
R_Trig_instance(Clk := Trigger);
IF (R_Trig_instance.Q=TRUE)THEN
   NXObject.Index    := UINT#16#4002;
   NXObject.Subindex := USINT#1;
END_IF;

// Execute NX_ReadObj instruction.
IF (Trigger=TRUE) THEN
   ReadCondition := TRUE;
END_IF;

IF ( (NX_ReadObj_instance.Done=TRUE) OR (NX_ReadObj_instance.Error=TRUE) ) THEN
   ReadCondition := FALSE;
END_IF;

ReadGo := ReadCondition & _EC_MBXSlavTbl[10];
NX_ReadObj_instance(
   Execute   := ReadGo,
   UnitProxy := NXUnitProxy,
   Obj       := NXObject,
   TimeOut   := UINT#2000,
   ReadDat   := IoRefreshMethod);

// Processing after instruction execution.
IF (NX_ReadObj_instance.Done=TRUE) THEN
   // Processing after normal end.
   NormalEnd := NormalEnd + UINT#1;
ELSIF (NX_ReadObj_instance.Error=TRUE) THEN
   // Processing after error end.
   ErrorEnd := ErrorEnd + UINT#1;
END_IF;
```

IO-Link Communications Instruction

IOL_ReadObj

The IOL_ReadObj instruction reads data from IO-Link device objects.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
IOL_ReadObj	Read IO-Link Device Object	FB		IOL_ReadObj_instance(Execute, DevicePort, DeviceObj, RetryCfg, ReadDat, Done, Busy, Error, ErrorID, ErrorType, ReadSize);

Version Information

A CPU Unit with unit version 1.12 or later and Sysmac Studio version 1.16 or higher are required to use this instruction.

Variables

	Meaning	I/O	Description	Valid range	Unit	Default
DevicePort	Device port	Input	Object that represents a device port	---	---	---
DeviceObj	IO-Link device object parameter		Specification for the IO-Link device object	---	---	---
RetryCfg	Execution retry setting		Setting for the instruction execution retry	---	---	---
ReadDat	Read data	In-out	Data read from IO-Link device	Depends on data type.	---	0
ErrorType	Error type	Output	Error code that is returned by IO-Link device is stored when *ErrorID* is 4800 hex.	16#0000 to 16#FFFF	---	---
ReadSize	Read data size		Size of data stored in *ReadDat*	10#1 to 10#232	Bytes	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
DevicePort	Refer to *Function* for details on the structure _sDEVICE_PORT.																			
DeviceObj	Refer to *Function* for details on the structure _sIOLOBJ_ACCESS.																			
RetryCfg	Refer to *Function* for details on the structure _sIOL_RETRY_CFG.																			
ReadDat	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	An array can also be specified.																			
ErrorType			OK																	
ReadSize							OK													

Function

The IOL_ReadObj instruction reads object data from IO-Link devices.

For the *DevicePort* input variable, set the IO-Link master unit and the port number to which the target IO-Link device for reading is connected.

The data type of the *DevicePort* input variable is structure _sDEVICE_PORT. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
DevicePort	Device port	Object that represents a device port	_sDEVICE_PORT	---	---	---
DeviceType	Device type	Type of the device to specify	_eDEVICE_TYPE	_DeviceNXUnit _DeviceEcatSlave _DeviceOptionBoard	---	---
NxUnit	Specified Unit	NX Unit to control	_sNXUNIT_ID	---	---	---
EcatSlave	Specified slave	EtherCAT slave to control	_sECAT_ID	---	---	---
OptBoard	Specified Option Board	Option Board to control	_sOPTBOARD_ID	---	---	---
Reserved	Reserved	Reserved	---	---	---	---
PortNo	Port number	Port number 1: Port 1 2: Port 2 3: Port 3 4: Port 4 5: Port 5 6: Port 6 7: Port 7 8: Port 8	USINT	Depends on data type.	---	---

Use *DeviceType* to specify the device type. Specify _DeviceNXUnit for an NX type of IO-Link master unit and _DeviceEcatSlave for a GX type of IO-Link master unit. The variable used to specify the device is determined by the specified device type.

For this instruction, it is determined as follows:

To specify the NX type, use *NxUnit* to specify the device. In this case, *EcatSlave* is not used. To *NxUnit*, pass the device variable that is assigned to the device to specify.

To specify the GX type, use *EcatSlave* to specify the device. In this case, *NxUnit* is not used. To *EcatSlave*, pass the device variable that is assigned to the device to specify.

Use *PortNo* to set the port number to which the IO-Link device is connected.

The number of ports differs depending on the type of IO-Link master unit.

NX type: 1 to 4

GX type: 1 to 8

The data type of *DeviceType* is enumerated type _eDEVICE_TYPE.

The meanings of the enumerators of enumerated type _eDEVICE_TYPE are as follows:

Enumerator	Meaning
_DeviceNXUnit	NX Unit is specified.
_DeviceEcatSlave	EtherCAT slave is specified.

Use the *DeviceObj* input variable to specify the object parameter for the IO-Link device from which data is read.

The data type of the *DeviceObj* input variable is structure _sIOLOBJ_ACCESS. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
DeviceObj	IO-Link device object parameter	Specification for the IO-Link device object	_sIOLOBJ_-ACCESS	---	---	---
Index	Index	Index	UINT	Depends on data type.	---	---
Subindex	Subindex	Set 0 to read from the entire index.	USINT	Depends on data type.	---	---

Use the *RetryCfg* input variable to set retry processing for instruction execution.

The data type of *RetryCfg* is structure _sIOL_RETRY_CFG. The specifications are as follows:

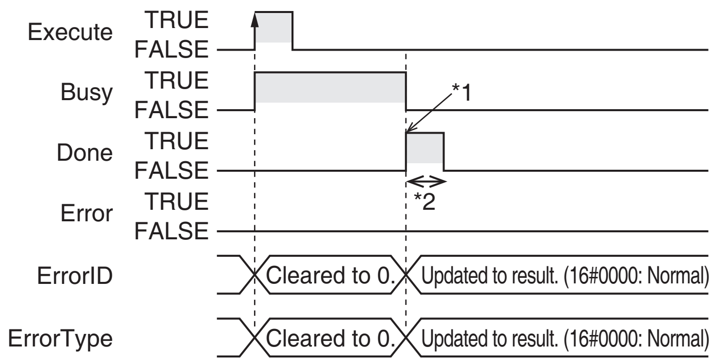
Name	Meaning	Description	Data type	Valid range	Unit	Default
RetryCfg	Execution retry setting	Setting for the instruction execution retry	_sIOL_RETRY_CFG	---	---	---
TimeOut	Timeout time	2.0 s when the timeout time is set to 0	TIME	0 to 300 s	---	T#2.0s
RetryNum	Number of retries	3 times if the number of retries at timeout is set to 0	UINT	Depends on data type.	Times	3

Data read from the IO-Link device is stored in the *ReadDat* in-out variable.

Timing Charts

The following figures show the timing charts.

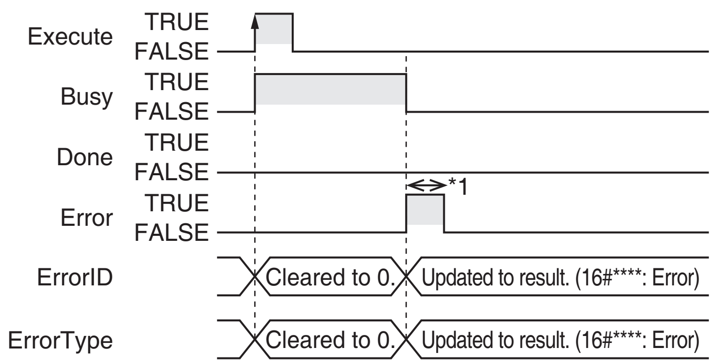
● Normal end



*1 Reading completed.

*2 Task period

● Error end



*1 Task period

Related System-defined Variables

Name	Meaning	Data type	Description
_EC_MBXSlavTbl	Message Communications Enabled Slave Table	ARRAY[1..512] OF BOOL*1	This table indicates the slaves that can perform message communications. Slaves are given in the table in the order of slave node addresses. TRUE: Communications are possible. FALSE: Communications are not possible.

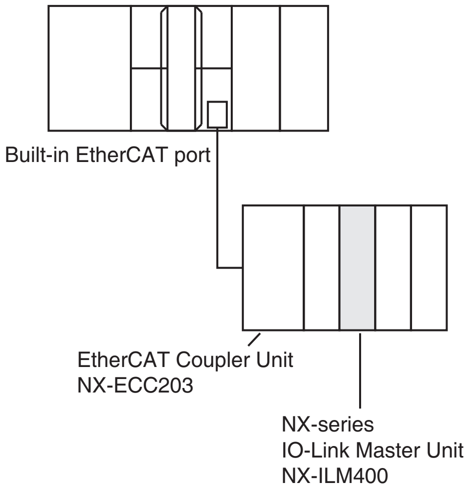
*1 The data type is ARRAY [1..192] OF BOOL for an NJ-series CPU Unit.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- For *DevicePort.NxUnit* and *DevicePort.EcatSlave*, specify the device variable that is assigned to the IO-Link master unit in the I/O Map of the Sysmac Studio. Refer to the *Sysmac Studio Version 1 Operation Manual* (Cat. No. W504-E1-07 or later) for details on assigning device variables.
- The size of the variable specified for *ReadDat* must be larger than the size of the object that is actually read.
- If *ReadDat* is STRING data, specify a variable whose size is the sum of the actually read string and a NULL character.
- If *ReadDat* is STRING data, the size that is output to *ReadSize* does not include the NULL character.
- Always use a variable for the parameter to pass to *ReadDat*. A building error will occur if a constant is passed.
- You can execute only one instruction at a time for the IO-Link master unit regardless of its type (NX or GX).
- You cannot use this instruction in an event task. A compiling error will occur.
- This instruction is executed when *Execute* changes to TRUE. The instruction is not executed when *Execute* is always TRUE.
- You can define a maximum of 64 instances for the IOL_ReadObj and IOL_WriteObj instructions.
- An error will occur in the following cases.
  - A value that is out of range was set for *DevicePort.NxUnit* or *DevicePort.EcatSlave*.
  - The size of the IO-Link device object to read is larger than the size of *ReadDat*. If this error occurs, the read data is not stored in *ReadDat*.
  - An error response was received from the IO-Link device.
    The upper eight bits represent *ErrorCode*, and lower eight bits represent *AdditionalCode*.
    For *ErrorCode* and *AdditionalCode*, refer to the Error type specifications of the IO-Link Communication Specification. You can obtain the Error type specifications from the IO-Link Consortium.
    http://www.io-link.com/
  - The specified IO-Link master unit does not exist.
  - The maximum number of messages that the IO-Link master can process is exceeded. Instruction execution is not possible because the IO-Link master is processing the messages from other applications.
  - The specified IO-Link master unit is not in a condition to receive messages.
  - More than 32 of the following instructions were executed at the same time: EC_CoESDOWrite, EC_CoESDORead, EC_StartMon, EC_StopMon, EC_SaveMon, EC_CopyMon, EC_DisconnectSlave, EC_ConnectSlave, EC_ChangeEnableSetting, IOL_ReadObj, and IOL_WriteObj.
  - A timeout occurred during communications.
  - The specified port of the IO-Link master unit is not the IO-Link mode. The port is disabled or in the SIO mode.
  - The IO-Link device is not connected to the specified port on the IO-Link master unit.
  - The IO power is not supplied to the specified port of the IO-Link master unit.
  - The specified port of the IO-Link master unit had a verification error or communications error.

Sample Programming

In this sample, an IO-Link master unit (NX-ILM400) is connected to an EtherCAT Coupler Unit (NX-ECC203).



The error log (Index:37/Subindex:0) of 30 bytes is read from the photoelectric sensor (E3Z) connected to port 1 on the NX-ILM400. The read data is stored in *DeviceErrorLog*.

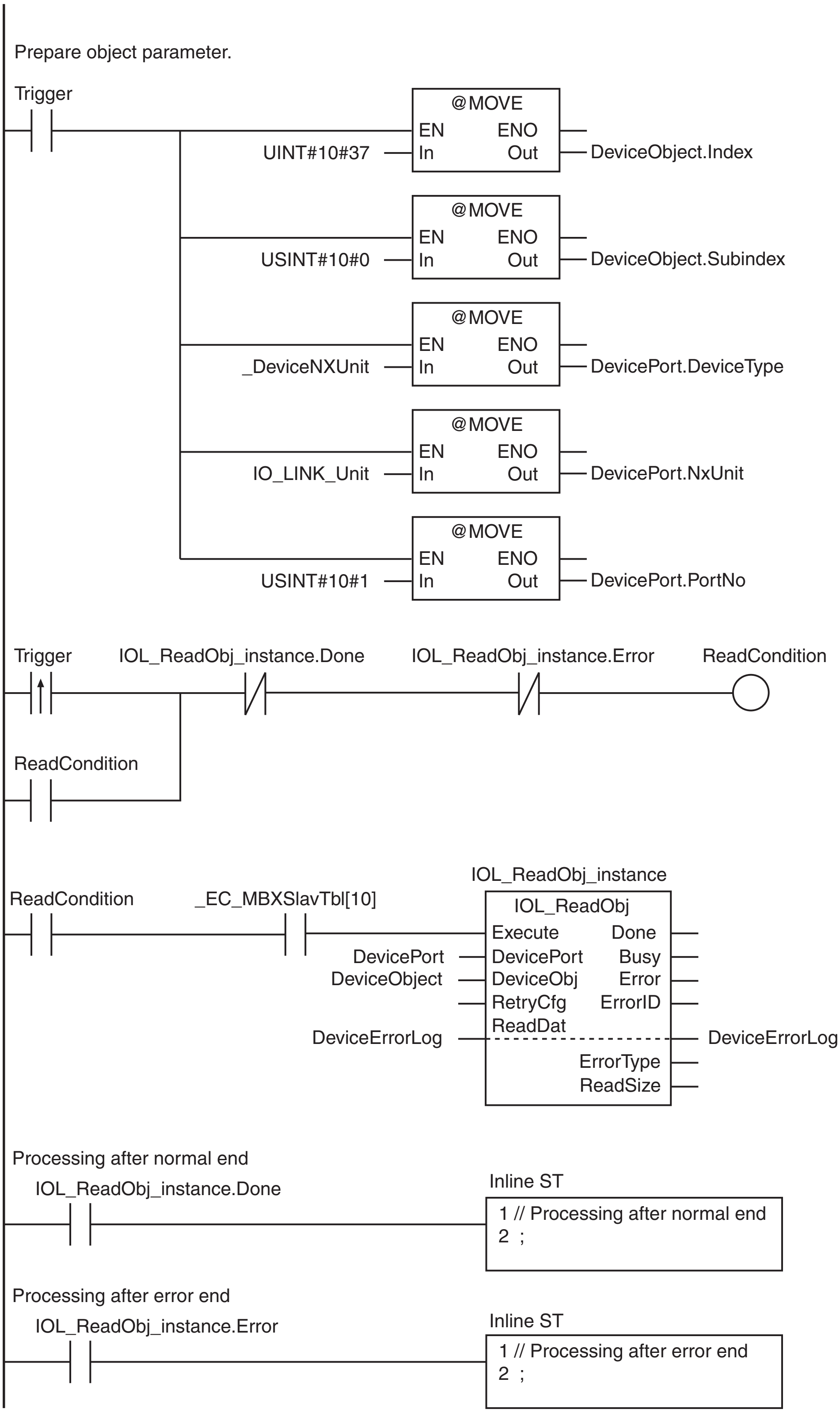
The node address of the NX-ECC203 is 10.

LD

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	FALSE	Execution condition
	ReadCondition	BOOL	FALSE	Data reading execution condition
	DevicePort	_sDEVICE_PORT		
	DeviceObject	_sIOLOBJ_ACCESS	(Index:=0, Subindex:=0)	Specification for the IO-Link device object
	DeviceErrorLog	ARRAY[1..30] OF BYTE		Read data
	IOL_ReadObj_instance	IOL_ReadObj		

External Variables	Variable	Constant	Initial value	Comment
	_EC_MBXSlavTbl	☑	ARRAY[1..512] OF BOOL*1	Message Communications Enabled Slave Table
	IO_LINK_Unit	☑	Set the device variable which specifies NX-ILM400 as the initial value of the structure member *NxUnit*.	

*1 For NJ-series CPU Units, the data type is ARRAY [1..192] OF BOOL.


Prepare object parameter.
Trigger
@MOVE
EN ENO
In Out
UINT#10#37
DeviceObject.Index
@MOVE
EN ENO
In Out
USINT#10#0
DeviceObject.Subindex
@MOVE
EN ENO
In Out
_DeviceNXUnit
DevicePort.DeviceType
@MOVE
EN ENO
In Out
IO_LINK_Unit
DevicePort.NxUnit
@MOVE
EN ENO
In Out
USINT#10#1
DevicePort.PortNo
Trigger
IOL_ReadObj_instance.Done
IOL_ReadObj_instance.Error
ReadCondition
ReadCondition
ReadCondition
_EC_MBXSlavTbl[10]
IOL_ReadObj_instance
IOL_ReadObj
Execute Done
DevicePort Busy
DeviceObj Error
RetryCfg ErrorID
ReadDat
ErrorType
ReadSize
DevicePort
DeviceObject
DeviceErrorLog
DeviceErrorLog
Processing after normal end
IOL_ReadObj_instance.Done
Inline ST
1 // Processing after normal end
2 ;
Processing after error end
IOL_ReadObj_instance.Error
Inline ST
1 // Processing after error end
2 ;

ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	FALSE	Execution condition
	ReadGo	BOOL	FALSE	Data reading execution
	DevicePort	_sDEVICE_PORT		
	DeviceObject	_sIOLOBJ_ACCESS	(Index:=0, Subindex:=0)	Specification for the IO-Link device object
	DeviceErrorLog	ARRAY[1..30] OF BYTE		Read data
	NormalEnd	UINT	0	Normal end
	ErrorEnd	UINT	0	Error end
	R_Trig_instance	R_Trig		
	IOL_ReadObj_instance	IOL_ReadObj		

External Variables	Variable	Constant	Initial value	Comment
	_EC_MBXSlavTbl	✓	ARRAY[1..512] OF BOOL*1	Message Communications Enabled Slave Table
	IO_LINK_Unit	✓	Set the device variable which specifies NX-ILM400 as the initial value of the structure member *NxUnit*.	

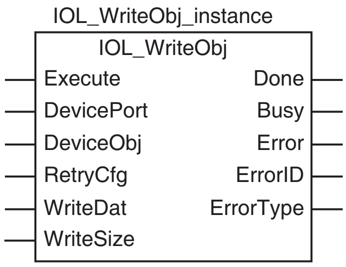
*1 For NJ-series CPU Units, the data type is ARRAY [1..192] OF BOOL.

```
// Prepare object parameter.
R_Trig_instance(Clk := Trigger);
IF (R_Trig_instance.Q=TRUE)THEN
        DeviceObject.Index := UINT#10#37;
        DeviceObject.Subindex := USINT#0;
        DevicePort.DeviceType:= _eDEVICE_TYPE#_DeviceNXUnit;
        DevicePort.NxUnit:= IO_LINK_Unit;
        DevicePort.PortNo:= USINT#10#1;
        IF ( _EC_MBXSlavTbl[10] =TRUE)THEN
                ReadGo := TRUE;
        END_IF;
END_IF;

IF ( (IOL_ReadObj_instance.Done=TRUE) OR (IOL_ReadObj_instance.Error=TRUE) ) THEN
        ReadGo:= FALSE;
END_IF;

// Execute IOL_ReadObj instruction.
IOL_ReadObj_instance(
        Execute := ReadGo,
        DevicePort:= DevicePort,
        DeviceObj := DeviceObject,
        ReadDat :=DeviceErrorLog);

// Processing after instruction execution
IF (IOL_ReadObj_instance.Done=TRUE) THEN
        // Processing after normal end
        NormalEnd := NormalEnd + UINT#1;
ELSIF (IOL_ReadObj_instance.Error=TRUE) THEN
```

```
        // Processing after error end
        ErrorEnd := ErrorEnd + UINT#1;
END_IF;
```

IOL_WriteObj

The IOL_WriteObj instruction writes data to IO-Link device objects.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
IOL_WriteObj	Write IO-Link Device Object	FB		IOL_WriteObj_instance(Execute, DevicePort, DeviceObj, RetryCfg, WriteDat, WriteSize, Done, Busy, Error, ErrorID, ErrorType);

Version Information

A CPU Unit with unit version 1.12 or later and Sysmac Studio version 1.16 or higher are required to use this instruction.

Variables

	Meaning	I/O	Description	Valid range	Unit	Default
DevicePort	Device port	Input	Object that represents a device port	---	---	---
DeviceObj	IO-Link device object parameter		Specification for the IO-Link device object	---	---	---
RetryCfg	Execution retry setting		Setting for the instruction execution retry	---	---	---
WriteDat	Write data		Data written to IO-Link device	Depends on data type.	---	---
WriteSize	Write data size		Write data size*1	10#1 to 10#232	Bytes	---
ErrorType	Error type	Output	Error code that is returned by IO-Link device is stored when *ErrorID* is 4800 hex.	16#0000 to 16#FFFF	---	---

*1 Input 1 if the written data is a BOOL data. Input the number of elements if the written data is a BOOL array.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
DevicePort	Refer to *Function* for details on the structure _sDEVICE_PORT.																			
DeviceObj	Refer to *Function* for details on the structure _sIOLOBJ_ACCESS.																			
RetryCfg	Refer to *Function* for details on the structure _sIOL_RETRY_CFG.																			
WriteDat	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	An array can also be specified.																			
WriteSize							OK													
ErrorType			OK																	

Function

The IOL_WriteObj instruction writes object data to IO-Link devices.

For the *DevicePort* input variable, set the IO-Link master unit and the port number to which the target IO-Link device for writing is connected.

The data type of the *DevicePort* input variable is structure _sDEVICE_PORT. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
DevicePort	Device port	Object that represents a device port	_sDEVICE_PORT	---	---	---
DeviceType	Device type	Type of the device to specify	_eDEVICE_TYPE	_DeviceNXUnit _DeviceEcatSlave _DeviceOptionBoard	---	---
NxUnit	Specified Unit	NX Unit to control	_sNXUNIT_ID	---	---	---
EcatSlave	Specified slave	EtherCAT slave to control	_sECAT_ID	---	---	---
OptBoard	Specified Option Board	Option Board to control	_sOPTBOARD_ID	---	---	---
Reserved	Reserved	Reserved	---	---	---	---
PortNo	Port number	Port number 1: Port 1 2: Port 2 3: Port 3 4: Port 4 5: Port 5 6: Port 6 7: Port 7 8: Port 8	USINT	Depends on data type.	---	---

Use *DeviceType* to specify the device type. Specify _DeviceNXUnit for an NX type of IO-Link master unit and _DeviceEcatSlave for a GX type of IO-Link master unit. The variable used to specify the device is determined by the specified device type.

For this instruction, it is determined as follows:

To specify the NX type, use *NxUnit* to specify the device. In this case, *EcatSlave* is not used. To *NxUnit*, pass the device variable that is assigned to the device to specify.

To specify the GX type, use *EcatSlave* to specify the device. In this case, *NxUnit* is not used. To *EcatSlave*, pass the device variable that is assigned to the device to specify.

Use *PortNo* to set the port number to which the IO-Link device is connected.

The number of ports differs depending on the type of IO-Link master unit.

NX type: 1 to 4

GX type: 1 to 8

The data type of *DeviceType* is enumerated type _eDEVICE_TYPE.

The meanings of the enumerators of enumerated type _eDEVICE_TYPE are as follows:

Enumerator	Meaning
_DeviceNXUnit	NX Unit is specified.
_DeviceEcatSlave	EtherCAT slave is specified.

Use the *DeviceObj* input variable to specify the object parameter for the IO-Link device to which data is written. The data type of the *DeviceObj* input variable is structure _sIOLOBJ_ACCESS. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
DeviceObj	IO-Link device object parameter	Specification for the IO-Link device object	_sIOLOBJ_-ACCESS	---	---	---
Index	Index	Index	UINT	Depends on data type.	---	---
Subindex	Subindex	Set 0 to read from the entire index.	USINT	Depends on data type.	---	---

Use the *RetryCfg* input variable to set retry processing for instruction execution.

The data type of *RetryCfg* is structure _sIOL_RETRY_CFG. The specifications are as follows:

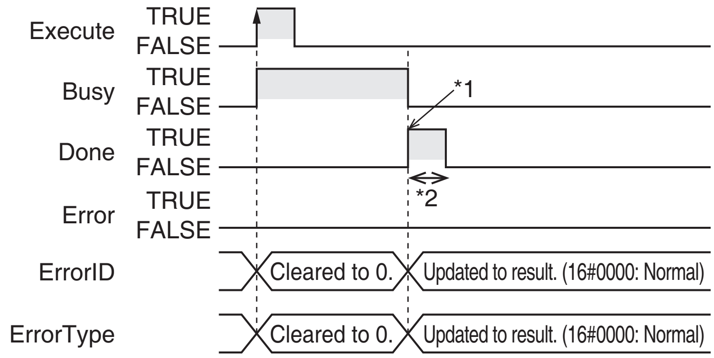
Name	Meaning	Description	Data type	Valid range	Unit	Default
RetryCfg	Execution retry setting	Setting for the instruction execution retry	_sIOL_RE-TRY_CFG	---	---	---
TimeOut	Timeout time	2.0 s when the timeout time is set to 0	TIME	0 to 300 s	---	T#2.0s
RetryNum	Number of retries	3 times if the number of retries at timeout is set to 0	UINT	Depends on data type.	Times	3

Use the *WriteDat* input variable to specify the data to write to the IO-Link device.

Timing Charts

The following figures show the timing charts.

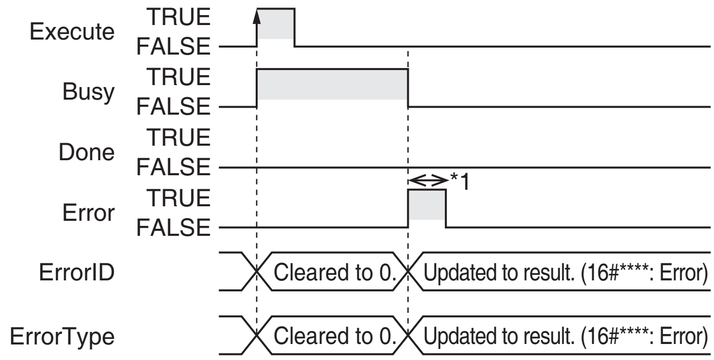
● Normal end



*1 Writing completed.

*2 Task period

● Error end



*1 Task period

Related System-defined Variables

Name	Name	Data type	Description
_EC_MBXSlavTbl	Message Communications Enabled Slave Table	ARRAY[1..512] OF BOOL*1	This table indicates the slaves that can perform message communications. Slaves are given in the table in the order of slave node addresses. TRUE: Communications are possible. FALSE: Communications are not possible.

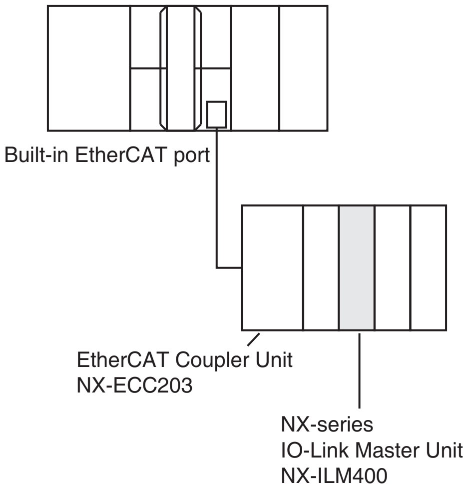
*1 The data type is ARRAY [1..192] OF BOOL for an NJ-series CPU Unit.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- For *DevicePort.NxUnit* and *DevicePort.EcatSlave*, specify the device variable that is assigned to the IO-Link master unit in the I/O Map of the Sysmac Studio. Refer to the *Sysmac Studio Version 1 Operation Manual* (Cat. No. W504-E1-07 or later) for details on assigning device variables.
- Always use a variable for the parameter to pass to *WriteDat*. A building error will occur if a constant is passed.
- You can execute only one instruction at a time for the IO-Link master unit regardless of its type (NX or GX).
- You cannot use this instruction in an event task. A compiling error will occur.
- This instruction is executed when *Execute* changes to TRUE. The instruction is not executed when *Execute* is always TRUE.
- You can define a maximum of 64 instances for the IOL_ReadObj and IOL_WriteObj instructions.
- An error will occur in the following cases.
  - A value that is out of range was set for *DevicePort.NxUnit* or *DevicePort.EcatSlave*.
  - The value of *TimeOut* is outside of the valid range.
  - The data type of *DevicePort* is invalid.
  - More than 232 bytes of data was specified for *WriteDat*.
  - An error response was received from the IO-Link device.
    The upper eight bits represent *ErrorCode*, and lower eight bits represent *AdditionalCode*.
    For *ErrorCode* and *AdditionalCode*, refer to the Error type specifications of the *IO-Link Communication Specification*. You can obtain the Error type specifications from the IO-Link Consortium.
    http://www.io-link.com/
  - The specified IO-Link master unit does not exist.
  - The maximum number of messages that the IO-Link master can process is exceeded. Instruction execution is not possible because the IO-Link master is processing the messages from other applications.
  - The specified IO-Link master unit is not in a condition to receive messages.
  - More than 32 of the following instructions were executed at the same time: EC_CoESDOWrite, EC_CoESDORead, EC_StartMon, EC_StopMon, EC_SaveMon, EC_CopyMon, EC_DisconnectSlave, EC_ConnectSlave, EC_ChangeEnableSetting, IOL_ReadObj, and IOL_WriteObj.
  - A timeout occurred during communications.
  - The specified port of the IO-Link master unit is not the IO-Link mode. The port is disabled or in the SIO mode.
  - The IO-Link device is not connected to the specified port on the IO-Link master unit.
  - The IO power is not supplied to the specified port of the IO-Link master unit.
  - The specified port of the IO-Link master unit had a verification error or communications error.

Sample Programming

In this sample, an IO-Link master unit (NX-ILM400) is connected to an EtherCAT Coupler Unit (NX-ECC203).



The value 01 is written to the one-byte SwitchPoint Logic Output 1 (Index: 61/Subindex: 1) of the photo-electric sensor (E3Z) connected to port 1 on the NX-ILM400. The written data is stored in *SwitchPoint*.

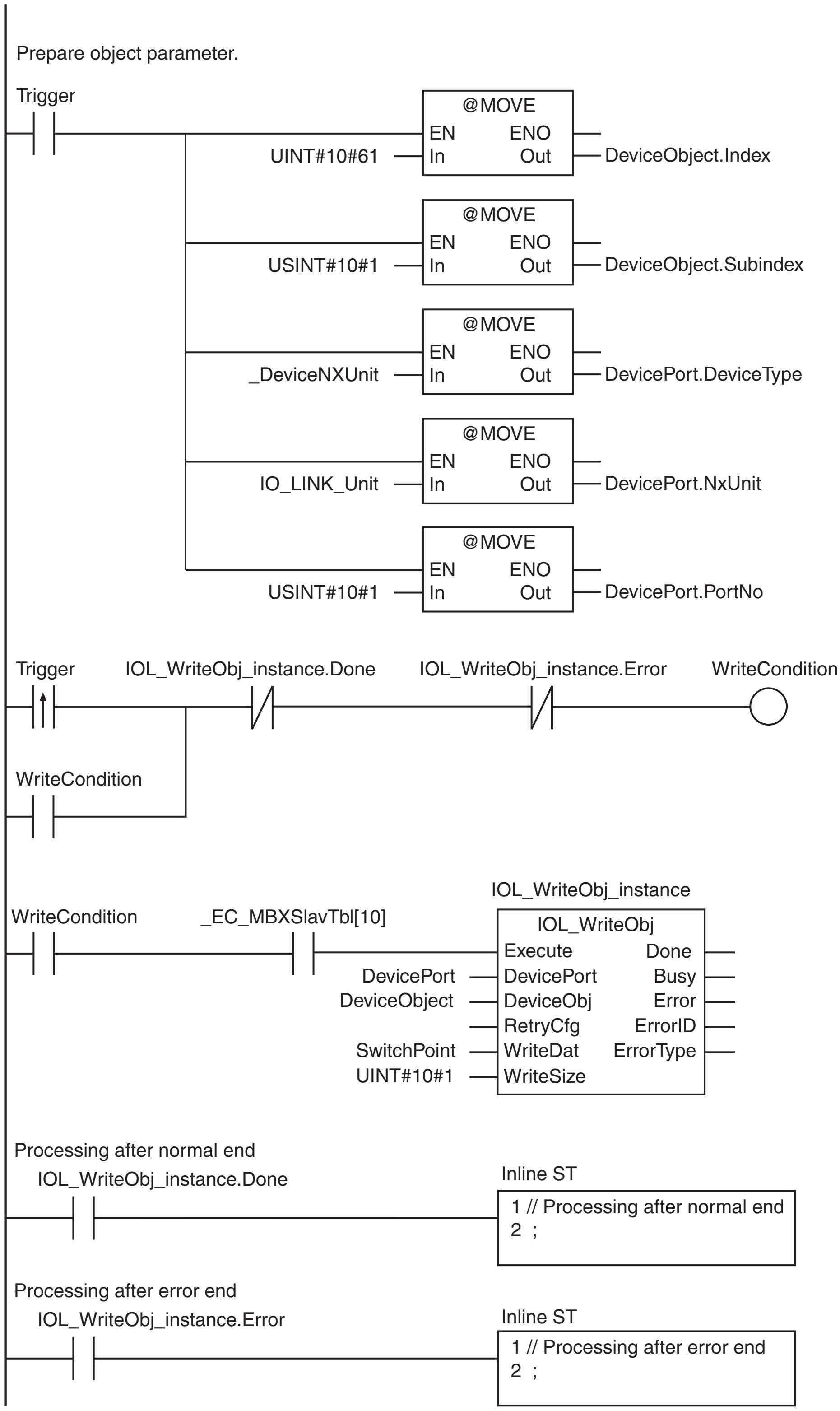
The node address of the NX-ECC203 is 10.

LD

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	FALSE	Execution condition
	WriteCondition	BOOL	FALSE	Data writing execution condition
	DevicePort	_sDEVICE_PORT		
	DeviceObject	_sIOLOBJ_ACCESS	(Index:=0, Sub-index:=0)	Specification for the IO-Link device object
	SwitchPoint	USINT	USINT#01	Write data
	IOL_WriteObj_instance	IOL_WriteObj		

External Variables	Variable	Constant	Initial value	Comment
	_EC_MBXSlavTbl	☑	ARRAY[1..512] OF BOOL*1	Message Communications Enabled Slave Table
	IO_LINK_Unit	☑	Set the device variable which specifies NX-ILM400 as the initial value of the structure member *NxUnit*.	

*1 For NJ-series CPU Units, the data type is ARRAY [1..192] OF BOOL.


Prepare object parameter.
Trigger
@MOVE
EN ENO
In Out
UINT#10#61
DeviceObject.Index
@MOVE
USINT#10#1
DeviceObject.Subindex
@MOVE
_DeviceNXUnit
DevicePort.DeviceType
@MOVE
IO_LINK_Unit
DevicePort.NxUnit
@MOVE
USINT#10#1
DevicePort.PortNo
Trigger
IOL_WriteObj_instance.Done
IOL_WriteObj_instance.Error
WriteCondition
WriteCondition
WriteCondition
_EC_MBXSlavTbl[10]
IOL_WriteObj_instance
IOL_WriteObj
Execute Done
DevicePort Busy
DeviceObj Error
RetryCfg ErrorID
WriteDat ErrorType
WriteSize
DevicePort
DeviceObject
SwitchPoint
UINT#10#1
Processing after normal end
IOL_WriteObj_instance.Done
Inline ST
1 // Processing after normal end
2 ;
Processing after error end
IOL_WriteObj_instance.Error
Inline ST
1 // Processing after error end
2 ;

ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	FALSE	Execution condition
	WriteGo	BOOL	FALSE	Data writing execution
	DevicePort	_sDEVICE_PORT		
	DeviceObject	_sIOLOBJ_ACCESS	(Index:=0, Subindex:=0)	Specification for the IO-Link device object
	SwitchPoint	USINT	USINT#01	Write data
	NormalEnd	UINT	0	Normal end
	ErrorEnd	UINT	0	Error end
	R_Trig_instance	R_Trig		
	IOL_WriteObj_instance	IOL_WriteObj		

External Variables	Variable	Constant	Initial value	Comment
	_EC_MBXSlavTbl	☑	ARRAY[1..512] OF BOOL*1	Message Communications Enabled Slave Table
	IO_LINK_Unit	☑	Set the device variable which specifies NX-ILM400 as the initial value of the structure member *NxUnit*.	

*1 For NJ-series CPU Units, the data type is ARRAY [1..192] OF BOOL.

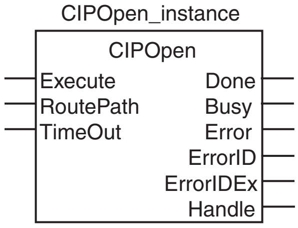
```
// Prepare object parameter.
R_Trig_instance(Clk := Trigger);
IF (R_Trig_instance.Q=TRUE)THEN
        DeviceObject.Index := UINT#10#61;
        DeviceObject.Subindex := USINT#1;
        DevicePort.DeviceType:= _eDEVICE_TYPE#_DeviceNXUnit;
        DevicePort.NxUnit:= IO_LINK_Unit;
        DevicePort.PortNo:= USINT#10#1;
        IF ( _EC_MBXSlavTbl[10] =TRUE)THEN
                WriteGo := TRUE;
        END_IF;
END_IF;

IF ( (IOL_WriteObj_instance.Done=TRUE) OR (IOL_WriteObj_instance.Error=TRUE) ) THEN
        WriteGo := FALSE;
END_IF;

// Execute IOL_WriteObj instruction.
IOL_WriteObj_instance(
                Execute := WriteGo,
                DevicePort:= DevicePort,
                DeviceObj := DeviceObject,
                WriteDat := SwitchPoint,
                WriteSize := UINT#10#1);

// Processing after instruction execution
IF (IOL_WriteObj_instance.Done=TRUE) THEN
        // Processing after normal end
        NormalEnd := NormalEnd + UINT#1;
```

```
ELSIF (IOL_WriteObj_instance.Error=TRUE) THEN
        // Processing after error end
        ErrorEnd := ErrorEnd + UINT#1;
END_IF;
```

EtherNet/IP Communications Instructions

CIPOpen

Opens a CIP class 3 connection (Large_Forward_Open) with the specified remote node. The data length is set to 1,994 bytes.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
CIPOpen	Open CIP Class 3 Connection (Large_Forward_Open)	FB		CIPOpen_instance(Execute, RoutePath, TimeOut, Done, Busy, Error, ErrorID, ErrorIDEx, Handle);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
RoutePath	Route path	Input	Route path	Depends on data type.	---	---
TimeOut	Timeout time	Input	Timeout time	1 to 65535	0.1 s	20 (2 s)
Handle	Handle	Output	Handle	---	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
RoutePath																				OK
TimeOut							OK													
Handle	Refer to *Function* for details on the structure _sCIP_HANDLE.																			

Function

The CIPOpen instruction opens a CIP class 3 connection (Large_Forward_Open) with a remote node on a CIP network. The remote node is specified with route path *RoutePath*. The data length is set to 1,994 bytes. The handle *Handle* is output when the connection is open.

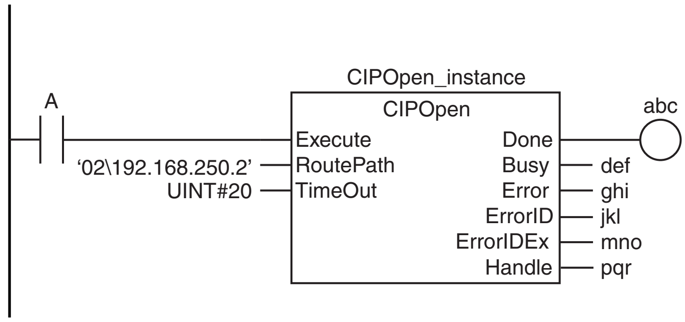
TimeOut specifies the connection timeout time. If a response does not return from the remote node within the connection timeout time after the CIPSend, CIPWrite, or CIPRead instruction is executed, it is assumed that communications failed. The connection timeout time is reset when the CIPRead, CIPWrite, or CIPSend instruction is executed and the remote node returns a response.

The data type of *Handle* is structure _sCIP_HANDLE. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Handle	Handle	Handle	_sCIP_HANDLE	---	---	---
Handle	Handle	Handle	UDINT	Depends on data type.	---	---

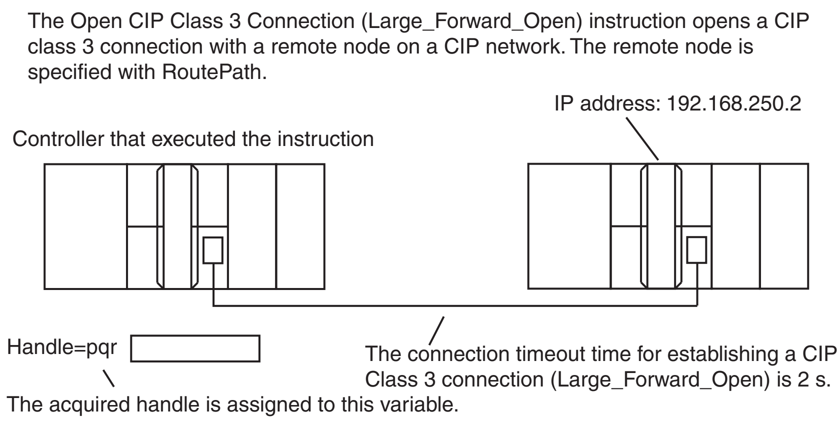
The following example is for when *RoutePath* is ‘02\192.168.250.2’ and *TimeOut* is UINT#20.
The Open CIP Class 3 Connection (Large_Forward_Open) instruction opens a CIP class 3 connection with the remote node with an IP address of 192.168.250.2. The timeout time is 2 s. The handle is assigned to variable *pqr*.

LD



ST

```
CIPOpen_instance(A, ‘2\192.168.250.2’, UINT#20, abc,
    def, ghi, jkl, mno, pqr);
```



If the value of *ErrorID* is WORD#16#1C00, the CIP message error code is stored in *ErrorIDEx*. The meaning and values of *ErrorIDEx* depend on the remote node. Refer to the manual for the remote node.

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta[*1]	Online	BOOL	This variable indicates when built-in EtherNet/IP port communications can be used. TRUE: Communications are possible. FALSE: Communications are not possible.
_EIP1_EtnOnlineSta[*2]			
_EIP2_EtnOnlineSta[*3]			
_EIPIn1_EtnOnlineSta[*4]			

*1 Use this variable name for an NJ-series CPU Unit.
*2 Use this variable name for port 1 on an NX-series CPU Unit, or for an NY-series Controller.
*3 Use this variable name for port 2 on an NX-series CPU Unit.
*4 Use this variable name for the internal communication port on an NY-series Controller.

Additional Information

- Refer to the following manuals for details on CIP communications.
  - *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User’s Manual* (Cat. No. W506)
  - *NY-series Industrial Panel PC / Industrial Box PC Built-in EtherNet/IP Port User’s Manual* (Cat. No. W563)

- *CJ-series EtherNet/IP Units Operation Manual for NJ-series CPU Unit* (Cat. No. W495)
- To establish a Forward Open connection or a connection with any given data length, use *CIPOpen-WithDataSize* on page 2-1007.

Version Information

A CPU Unit with unit version 1.06 or later and Sysmac Studio version 1.07 or higher are required to use the CIPOpenWithDataSize instruction.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- You must execute this instruction or the CIPOpenWithDataSize instruction before you execute the CIPRead, CIPWrite, or CIPSend instruction.
- For this instruction, the first timeout time after a connection is established is 10 s even if the value of *TimeOut* is set to less than 100 (10 s).
- Use the CIPClose instruction to close connections that were opened with the CIPOpen instruction.
- Even if the connection times out, the handle created by this instruction will remain. Always use the CIPClose instruction to close the connection.
- Handles that are created with this instruction are disabled when you change to PROGRAM mode.
- You can create a maximum of 32 handles at the same time.
- You can use this instruction only through an NJ/NX-series CPU Unit, through a built-in EtherNet/IP port on an NY-series Controller, or through an EtherNet/IP Unit connected to an NJ-series CPU Unit.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The value of *TimeOut* is outside of the valid range.
  - The text string in *RoutePath* is not valid.
  - More than 32 CIP-related instructions were executed simultaneously.
  - An attempt was made to open a connection beyond the CIPClass connection resources (32 connections).
  - A connection opened response was not received.
  - The remote node to which to open a connection does not support Large_Forward_Open.
  - There is a setting error for the local IP address.
  - A duplicated IP error occurred.
  - All TCP connections are already in use.
  - The instruction was executed when there was a BOOTP server error.

Version Information

For CPU Unit version 1.10 or later, the value of *Handle* does not change even if *Error* changes to TRUE. For version 1.09 or earlier, the value of *Handle* changes to 0.

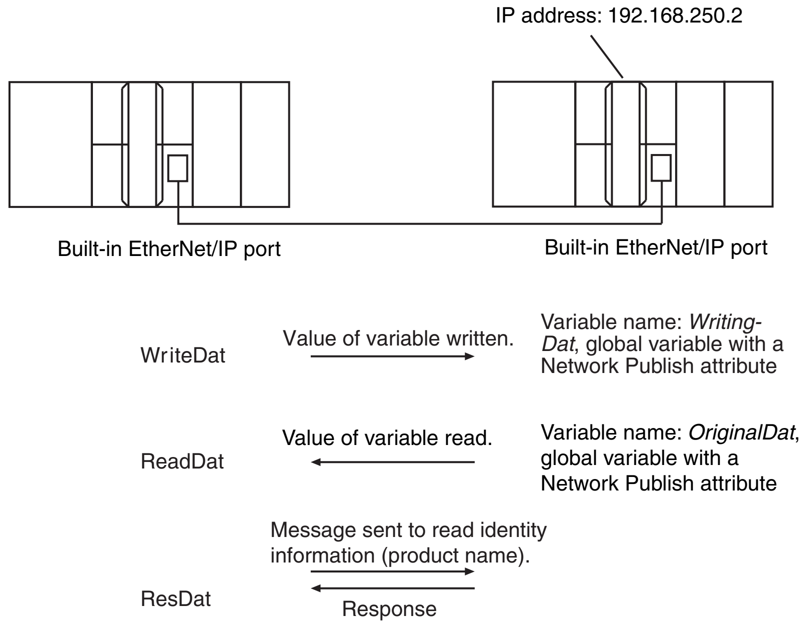
Sample Programming

This sample uses CIP class 3 messages to write a variable, read a variable, and send a message. The Controllers are connected to an EtherNet/IP network. The IP address of the remote node is 192.168.250.2.
The following procedure is used.

1. The CIPOpen is used to open a class 3 connection (Large_Forward_Open). The timeout time is 2 s.
2. The CIPWrite instruction is used to write the value of a variable at a remote node. The variable name at the remote node is *WritingDat* and the contents of the *WriteDat* is written to it. *Writing-Dat* must be defined as a global variable at the remote node and the Network Publish attribute must be set.
3. The CIPRead instruction is used to read the value of a variable at a remote node. The value of the variable *OriginalDat* at the other node is read and the read value is stored in the *ReadDat* variable. *OriginalDat* must be defined as a global variable at the remote node and the Network Publish attribute must be set.
4. The CIPSend instruction is used to send an explicit message to a remote node. The contents of the message is to read identity information (product name). The class ID, instance ID, attribute ID, and service code are as follows: The response data is stored in the *ResDat* variable.

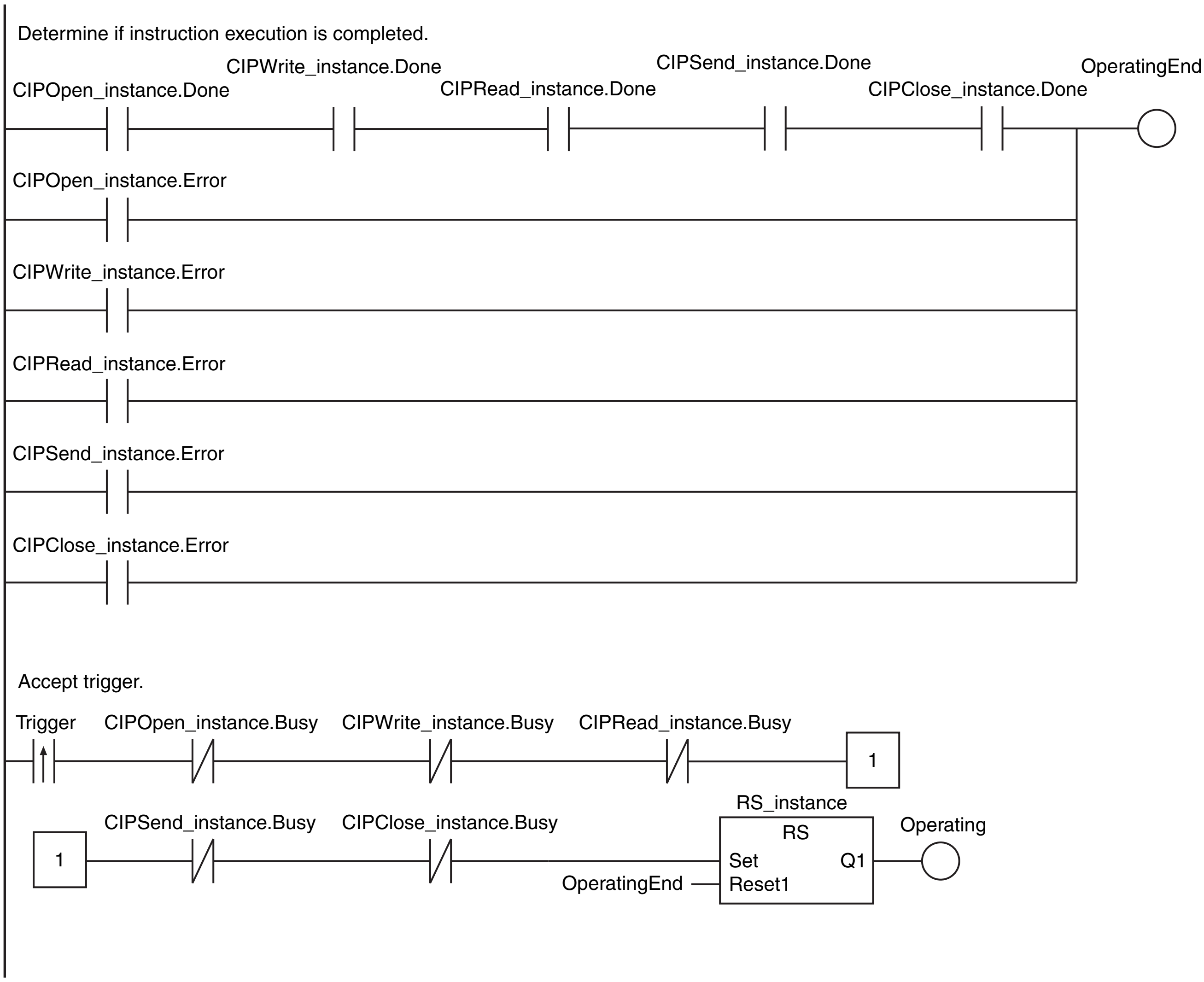
Item	Value
Class ID	1
Instance ID	1
Attribute ID	7
Service code	16#0E

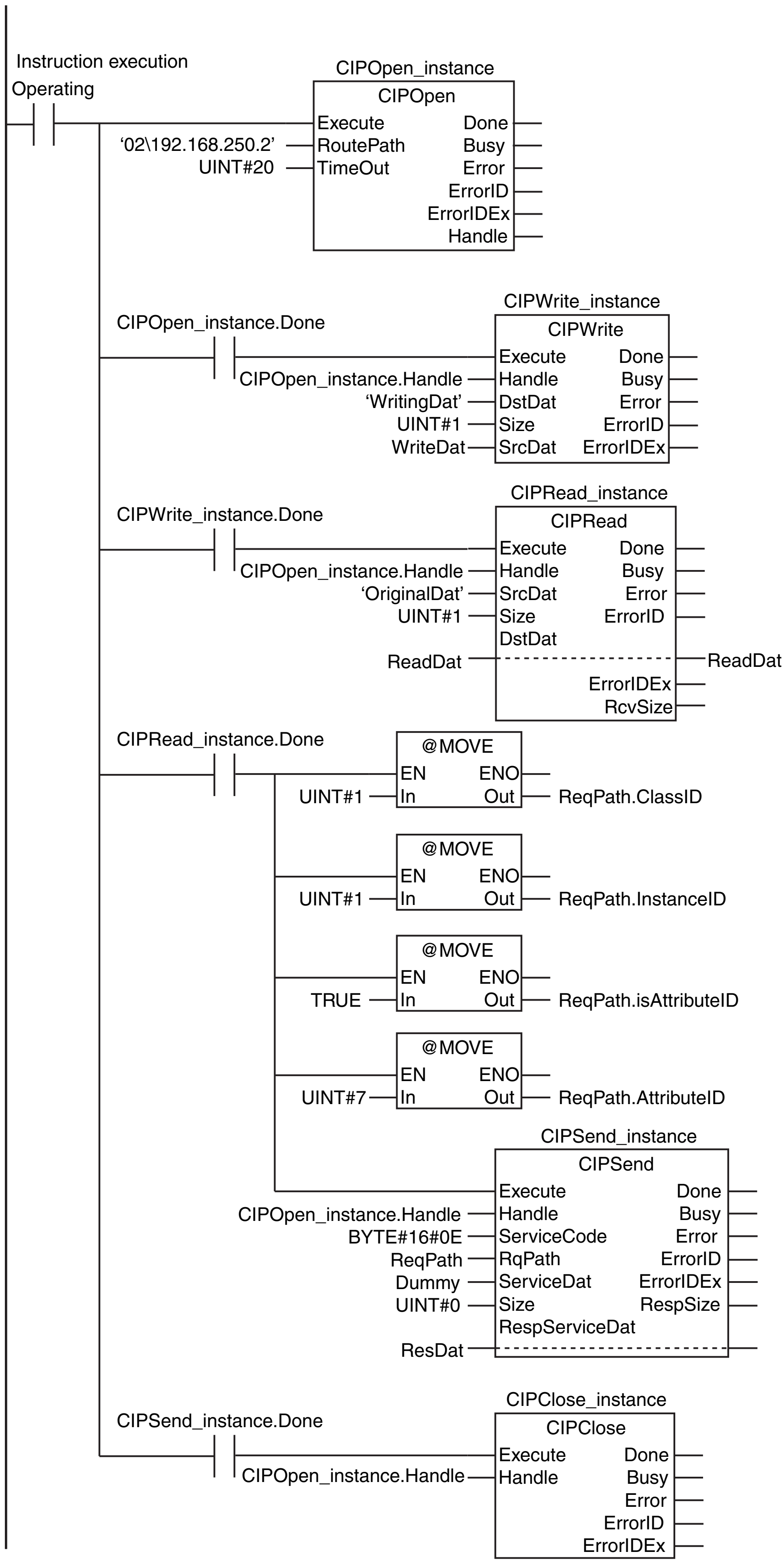
5. The CIPClose instruction is used to close the class 3 connection.

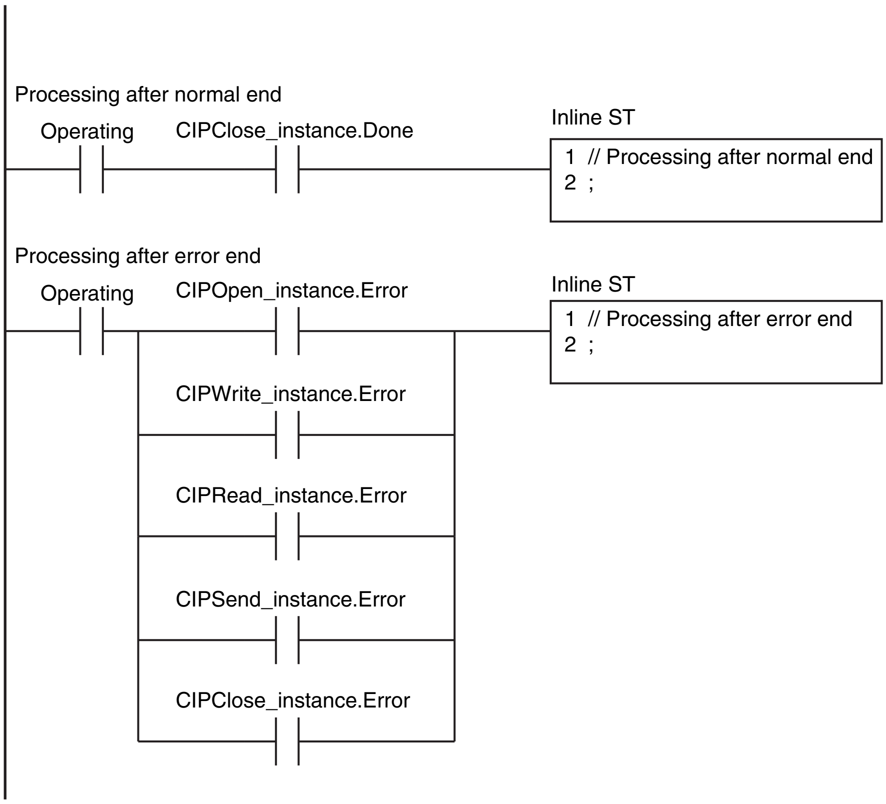


LD

Variable	Data type	Initial value	Comment
OperatingEnd	BOOL	FALSE	Processing completed
Trigger	BOOL	FALSE	Execution condition
Operating	BOOL	FALSE	Processing
WriteDat	INT	1234	Write data
ReadDat	INT	0	Read data
ReqPath	_sREQUEST_PATH	(ClassID:=0, InstanceID:=0, isAttributeID:=FALSE, AttributeID:=0)	Request path
ResDat	ARRAY[0..10] OF BYTE	[11(16#0)]	Response data
Dummy	BYTE	16#0	Dummy
RS_instance	RS		
CIPOpen_instance	CIPOpen		
CIPWrite_instance	CIPWrite		
CIPRead_instance	CIPRead		
CIPSend_instance	CIPSend		
CIPClose_instance	CIPClose		




Instruction execution
Operating
CIPOpen_instance
CIPOpen
Execute Done
'02\192.168.250.2' RoutePath Busy
UINT#20 TimeOut Error
ErrorID
ErrorIDEx
Handle
CIPOpen_instance.Done
CIPWrite_instance
CIPWrite
Execute Done
CIPOpen_instance.Handle Handle Busy
'WritingDat' DstDat Error
UINT#1 Size ErrorID
WriteDat SrcDat ErrorIDEx
CIPWrite_instance.Done
CIPRead_instance
CIPRead
Execute Done
CIPOpen_instance.Handle Handle Busy
'OriginalDat' SrcDat Error
UINT#1 Size ErrorID
DstDat
ReadDat ReadDat
ErrorIDEx
RcvSize
CIPRead_instance.Done
@MOVE
EN ENO
UINT#1 In Out ReqPath.ClassID
@MOVE
EN ENO
UINT#1 In Out ReqPath.InstanceID
@MOVE
EN ENO
TRUE In Out ReqPath.isAttributeID
@MOVE
EN ENO
UINT#7 In Out ReqPath.AttributeID
CIPSend_instance
CIPSend
Execute Done
CIPOpen_instance.Handle Handle Busy
BYTE#16#0E ServiceCode Error
ReqPath RqPath ErrorID
Dummy ServiceDat ErrorIDEx
UINT#0 Size RespSize
RespServiceDat
ResDat
CIPSend_instance.Done
CIPClose_instance
CIPClose
Execute Done
CIPOpen_instance.Handle Handle Busy
Error
ErrorID
ErrorIDEx


Processing after normal end
Operating
CIPClose_instance.Done
Inline ST
1 // Processing after normal end
2 ;
Processing after error end
Operating
CIPOpen_instance.Error
Inline ST
1 // Processing after error end
2 ;
CIPWrite_instance.Error
CIPRead_instance.Error
CIPSend_instance.Error
CIPClose_instance.Error

ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	FALSE	Execution condition
	DoCIPTrigger	BOOL	FALSE	Processing
	Stage	INT	0	Stage change
	WriteDat	INT	0	Write data
	ReadDat	INT	0	Read data
	ReqPath	_sREQUEST_PATH	(ClassID:=0, InstanceID:=0, isAttributeID:=FALSE, AttributeID:=0)	Request path
	ResDat	ARRAY[0..10] OF BYTE	[11(16#0)]	Response data
	Dummy	BYTE	16#0	Dummy
	CIPOpen_instance	CIPOpen		
	CIPWrite_instance	CIPWrite		
	CIPRead_instance	CIPRead		
	CIPSend_instance	CIPSend		
	CIPClose_instance	CIPClose		

External Variables	Variable	Constant	Data type	Comment
	_EIP_EtnOnlineSta	✔	BOOL	Online

```
// Start sequence when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (DoCIPTrigger=FALSE) AND (_Eip_EtnOnlineSta=TRUE) ) THEN
    DoCIPTrigger:=TRUE;
    Stage       :=INT#1;
    CIPOpen_instance(Execute :=FALSE);     // Initialize instance.
    CIPWrite_instance(
        Execute             :=FALSE,       // Initialize instance.
        SrcDat              :=WriteDat);   // Dummy
    CIPRead_instance(                      // Initialize instance.
        Execute             :=FALSE,       // Dummy
        DstDat              :=ReadDat);    // Dummy
    CIPSend_instance(
        Execute             :=FALSE,       // Initialize instance.
        ServiceDat          := Dummy,      // Dummy
        RespServiceDat      :=ResDat);     // Dummy
    CIPClose_instance(Execute:=FALSE);     // Initialize instance.
END_IF;


IF (DoCIPTrigger=TRUE) THEN
    CASE Stage OF
    1 :      // Open CIP Class 3 Connection (Large_Forward_Open)
        CIPOpen_instance(
            Execute   :=TRUE,
            TimeOut   :=UINT#20,             // Timeout time: 2.0 s
            RoutePath :='02\192.168.250.2'); // Route path

        IF (CIPOpen_instance.Done=TRUE) THEN
            Stage:=INT#2;                    // Normal end
        ELSIF (CIPOpen_instance.Error=TRUE) THEN
            Stage:=INT#10;                   // Error end
        END_IF;

    2 :      // Request writing value of variable.
        CIPWrite_instance(
            Execute :=TRUE,
```

```
            Handle  :=CIPOpen_instance.Handle,// Handle
            DstDat  :='WritingDat',           // Destination variable name
            Size    :=UINT#1,                 // Number of elements to write
            SrcDat  :=WriteDat);              // Write data

        IF (CIPWrite_instance.Done=TRUE) THEN
            Stage:=INT#3;                     // Normal end
        ELSIF (CIPWrite_instance.Error=TRUE) THEN
            Stage:=INT#20;                    // Error end
        END_IF;

    3 :      // Request reading value of variable.
        CIPRead_instance(
            Execute :=TRUE,
            Handle  :=CIPOpen_instance.Handle,// Handle
            SrcDat  :='OriginalDat',          // Destination variable name
            Size    :=UINT#1,                 // Number of elements to read
            DstDat  :=ReadDat);               // Read data

        IF (CIPRead_instance.Done=TRUE) THEN
            Stage:=INT#4;                     // Normal end
        ELSIF (CIPRead_instance.Error=TRUE) THEN
            Stage:=INT#30;                    // Error end
        END_IF;

    4 :      // Send message
        ReqPath.ClassID       :=UINT#01;
        ReqPath.InstanceID    :=UINT#01;
        ReqPath.isAttributeID :=TRUE;
        ReqPath.AttributeID   :=UINT#07;
        CIPSend_instance(
            Execute       :=TRUE,
            Handle        :=CIPOpen_instance.Handle,  // Handle
            ServiceCode   :=BYTE#16#0E,               // Service code
            RqPath        :=ReqPath,                  // Request path
            ServiceDat    :=Dummy,                    // Service data
            Size          :=UINT#0,                   // Number of elements
            RespServiceDat:=ResDat);                  // Response data

        IF (CIPSend_instance.Done=TRUE) THEN
            Stage:=INT#5;    // Normal end
        ELSIF (CIPSend_instance.Error=TRUE) THEN
            Stage:=INT#40;   // Error end
    END_IF;

    5 :      // Request closing CIP class 3 connection.
        CIPClose_instance(
            Execute :=TRUE,
            Handle  :=CIPOpen_instance.Handle);       // Handle

        IF (CIPClose_instance.Done=TRUE) THEN
            Stage:=INT#0;
        ELSIF (CIPClose_instance.Error=TRUE) THEN
            Stage:=INT#50;
        END_IF;

    0:      // Processing after normal end
        DoCIPTrigger:=FALSE;
        Trigger     :=FALSE;

    ELSE    // Processing after error end
        DoCIPTrigger:=FALSE;
        Trigger     :=FALSE;
    END_CASE;
END_IF;
```

CIPOpenWithDataSize

The CIPOpenWithDataSize instruction opens a CIP class 3 connection with the specified remote node that allows class 3 explicit messages of the specified data length or shorter to be sent and received.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
CIPOpenWith-DataSize	Open CIP Class 3 Connection with Specified Data Size	FB	CIPOpenWithDataSize_instance CIPOpenWithDataSize Execute — Done RoutePath — Busy TimeOut — Error DataSize — ErrorID ErrorIDEx Handle	CIPOpen_instance(Execute, RoutePath, TimeOut, DataSize, Done, Busy, Error, ErrorID, ErrorIDEx, Handle);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
RoutePath	Route path	Input	Route path	Depends on data type.	---	---
TimeOut	Timeout time		Timeout time	1 to 65,535	0.1 s	20 (2s)
DataSize	Data length		Data length	6 to 8,192*1*2	Bytes	1994
Handle	Handle	Output	Handle	---	---	---

*1 The range is 6 to 1,994 for NX1P2 and NJ-series CPU Units.

*2 With a CPU Unit with unit version 1.10 or earlier or Sysmac Studio version 1.14 or lower, the minimum value is 10.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
RoutePath																				OK
TimeOut							OK													
DataSize							OK													
Handle	Refer to *Function* for details on the structure _sCIP_HANDLE.																			

Function

The CIPOpenWithDataSize instruction opens a CIP class 3 connection with a remote node on a CIP network. The remote node is specified with route path *RoutePath*. Data length *DataSize* specifies the data length of class 3 explicit messages that can be sent and received.

The class 3 connection service is determined by the value of *DataSize* as given in the following table.

Value of *DataSize* [bytes]	Service
509 or less	Forward_Open
510 to 8,192*	Large_Forward_Open

* The range is 510 to 1,994 for NX1P2 and NJ-series CPU Units.

The handle *Handle* is output when the connection is open.

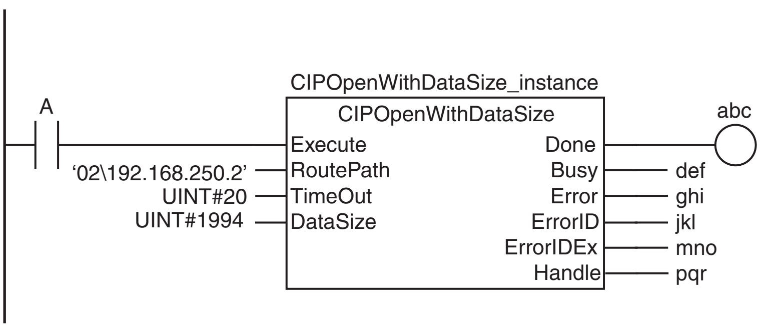
TimeOut specifies the connection timeout time. If a response does not return from the remote node within the connection timeout time after the CIPSend, CIPWrite, or CIPRead instruction is executed, it is assumed that communications failed. The connection timeout time is reset when the CIPRead, CIP-Write, or CIPSend instruction is executed and the remote node returns a response.

The data type of *Handle* is structure _sCIP_HANDLE. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Handle	Handle	Handle	_sCIP_ HANDLE	---	---	---
Handle	Handle	Handle	UDINT	Depends on data type.	---	---

The following example is for when *RoutePath* is ‘02\192.168.250.2’ and *TimeOut* is UINT#20. The CIPOpenWithDataSize instruction opens a CIP class 3 connection with the remote node with an IP address of 192.168.250.2. The data length is 1,994 bytes and the timeout time is 2 s. The handle is assigned to variable *pqr*.

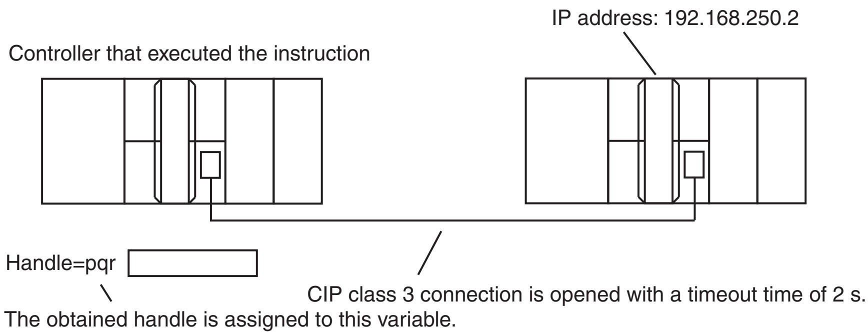
LD



ST

```
CIPOpenWithDataSize_instance(A,
                '2\192.168.250.2',
                UINT#20, UINT#1994,
                abc, def, ghi, jkl, mno,
                pqr);
```

The CIPOpenWithDataSize instruction opens a CIP class 3 connection with a remote node on a CIP network. The remote node is specified with *RoutePath*.



If the value of *ErrorID* is WORD#16#1C00, the CIP message error code is stored in *ErrorIDEx*. The meaning and values of *ErrorIDEx* depend on the remote node. Refer to the manual for the remote node.

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta*1 _EIP1_EtnOnlineSta*2 _EIP2_EtnOnlineSta*3 _EIPIn1_EtnOnlineSta*4	Online	BOOL	This variable indicates when built-in EtherNet/IP port communications can be used. TRUE: Communications are possible. FALSE: Communications are not possible.

*1 Use this variable name for an NJ-series CPU Unit.

*2 Use this variable name for port 1 on an NX-series CPU Unit, or for an NY-series Controller.

*3 Use this variable name for port 2 on an NX-series CPU Unit.

*4 Use this variable name for the internal communication port on an NY-series Controller.

Additional Information

- Refer to the following manuals for details on CIP communications.
  - *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User's Manual* (Cat. No. W506)
  - *NY-series Industrial Panel PC / Industrial Box PC Built-in EtherNet/IP Port User's Manual* (Cat. No. W563)
  - *CJ-series EtherNet/IP Units Operation Manual for NJ-series CPU Unit* (Cat. No. W495)
- To use Large_Forward_Open as the class 3 connection service, you can also use the CIPOpen instruction (page 2-998).

Precautions for Correct Use

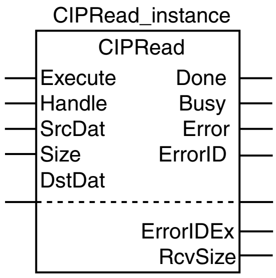
- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* (page 2-3) for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- You must execute this instruction or the CIPOpen instruction before you execute the CIPRead, CIPWrite, or CIPSend instruction.
- For this instruction, the first timeout time after a connection is established is 10 s even if the value of *TimeOut* is set to less than 100 (10 s).
- Use the CIPClose instruction to close connections that were opened with the CIPOpenWithDataSize instruction.
- Even if the connection times out, the handle created by this instruction will remain. Always use the CIPClose instruction to close the connection.
- Handles that are created with this instruction are disabled when you change to PROGRAM mode.
- You can create a maximum of 32 handles at the same time.
- You can use this instruction only through an NJ/NX-series CPU Unit, through a built-in EtherNet/IP port on an NY-series Controller, or through an EtherNet/IP Unit connected to an NJ-series CPU Unit.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The value of *TimeOut* is outside of the valid range.
  - The text string in *RoutePath* is not valid.
  - More than 32 CIP-related instructions were executed simultaneously.
  - An attempt was made to open a connection beyond the CIPClass connection resources (32 connections).
  - A connection opened response was not received.
  - The value of *DataSize* is 510 to 1,994 and the remote node to which to open a connection does not support Large_Forward_Open.
  - There is a setting error for the local IP address.
  - A duplicated IP error occurred.
  - All TCP connections are already in use.
  - The instruction was executed when there was a BOOTP server error.

Version Information

- A CPU Unit with unit version 1.06 or later and Sysmac Studio version 1.07 or higher are required to use this instruction.
- For CPU Unit version 1.10 or later, the value of *Handle* does not change even if *Error* changes to TRUE. For version 1.09 or earlier, the value of *Handle* changes to 0.

CIPRead

The CIPRead instruction uses a class 3 explicit message to read the value of a variable in another Controller on a CIP network.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
CIPRead	Read Variable Class 3 Explicit	FB		CIPRead_instance(Execute, Handle, SrcDat, Size, DstDat, Done, Busy, Error, ErrorID, ErrorIDEx, RcvSize);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Handle	Handle	Input	Handle obtained with CIPOpen or CIPOpenWithDataSize instruction	---	---	---
SrcDat	Source variable name		Name of variable to read in other Controller	Depends on data type.		''
Size	Number of elements to read		Number of elements to read	0 to 8,186*		1
DstDat	Read data	In-out	Read data value	Depends on data type.	---	---
RcvSize	Read data size	Output	Read data size	0 to 8,186*	Bytes	---

* The range is 0 to 1,988 for NX1P2 and NJ-series CPU Units.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Handle	Refer to *Function* for details on the structure _sCIP_HANDLE.																			
SrcDat																				OK
Size							OK													
DstDat	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	An enumeration, array, structure, structure member, or union member can also be specified.*																			
RcvSize							OK													

* You cannot specify a STRING array.

Function

The CIPRead instruction reads the value of the network variable specified with source variable name *SrcDat* from another Controller on a CIP network. The other Controller is specified with *Handle*.
The read data value is stored in *DstDat.*
Size specifies the number of elements to read. If *SrcDat* is an array, specify the number of elements to read with *Size*. If *SrcDat* is not an array, always specify 1 for *Size*. If the value of *Size* is 0, nothing is read regardless of whether *SrcDat* is an array or not.
When the read operation is completed, the number of bytes of the data that was read is assigned to read data size *RcvSize*.
The maximum size of the data that you can read depends on the instruction that established the connection and the data type of the data that is read as shown in the following table.

Instruction that established the connection	Data type of read data	Maximum size of data that you can read [bytes]
CIPOpen	Structure	1984
	STRING	1986
	Other data type	1988
CIPOpenWithDataSize	Structure	*DataSize* in CIPOpenWithDataSize instruction – 10
	STRING	*DataSize* in CIPOpenWithDataSize instruction – 8
	Other data type	*DataSize* in CIPOpenWithDataSize instruction – 6

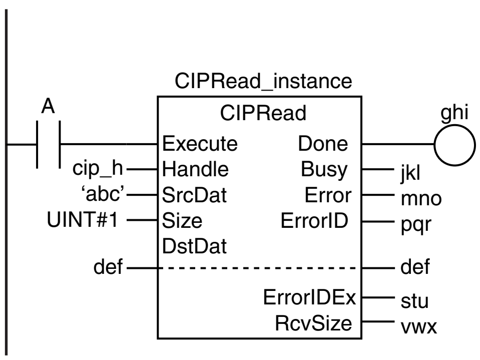
The data type of *Handle* is structure _sCIP_HANDLE. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Handle	Handle	Handle	_sCIP_ HANDLE	---	---	---
Handle	Handle	Handle	UDINT	Depends on data type.	---	---

If the value of *ErrorID* is WORD#16#1C00, the CIP message error code is stored in *ErrorIDEx*.

In the following example, the value of variable *abc* in the remote Controller is read and stored in the variable *def* in the local Controller. The number of elements to read *Size* is UINT#1. The data type of *abc* and *def* is SINT. The size of SINT data is one byte, so the value of the read data size *vwx* is UINT#1.

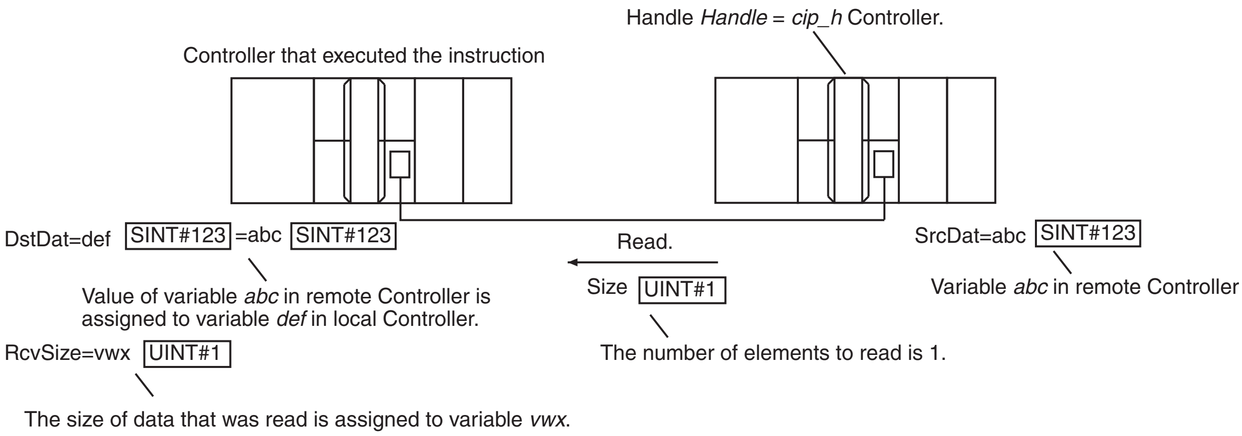
LD



ST

```
CIPRead_instance(A, cip_h, 'abc', UINT#1,
                 def, ghi, jkl, mno, pqr,
                 stu, vwx);
```

The value of variable *SrcDat* in remote Controller on the CIP network specified by the handle *Handle* is assigned to variable *DstDat* in the local Controller. *Size* specifies the number of elements to read. The size of data that was read is assigned to *RcvSize*.

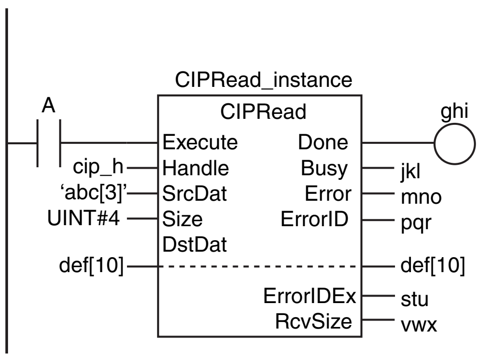


Reading Arrays

To read array data, pass a subscripted array element to *ScrDat* as the parameter. Also pass a subscripted array element to *DstDat* as the parameter.

The following example reads the four array variable elements *abc[3]* to *abc[6]* from the remote Controller and stores the results in array variable elements *def[10]* to *def[13]* in the local Controller. The data type of *abc* and *def* is INT. The size of INT data is two bytes, so the value of the read data size *vwx* is UINT#8.

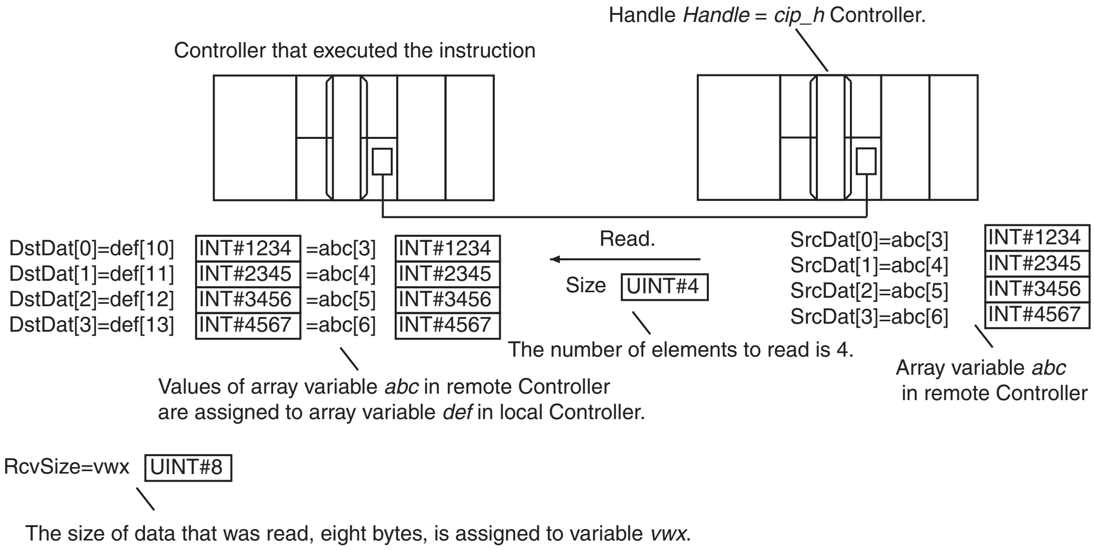
LD



ST

```
CIPRead_instance(A, cip_h, ‘abc[3]’, UINT#4,
                 def[10], ghi, jkl, mno, pqr,
                 stu, vwx);
```

Values of array variable elements *abc[3]* to *abc[6]* in remote Controller are assigned to array variable elements *def[10]* to *def[13]* in local Controller.



Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta*1	Online	BOOL	This variable indicates when built-in EtherNet/IP port communications can be used. TRUE: Communications are possible. FALSE: Communications are not possible.
_EIP1_EtnOnlineSta*2 _EIP2_EtnOnlineSta*3			
_EIPIn1_EtnOnlineSta*4			

*1 Use this variable name for an NJ-series CPU Unit.

*2 Use this variable name for port 1 on an NX-series CPU Unit, or for an NY-series Controller.

*3 Use this variable name for port 2 on an NX-series CPU Unit.

*4 Use this variable name for the internal communication port on an NY-series Controller.

Additional Information

Refer to the following manuals for details on CIP communications.

- *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User's Manual* (Cat. No. W506)
- *NY-series Industrial Panel PC / Industrial Box PC Built-in EtherNet/IP Port User's Manual* (Cat. No. W563)
- *CJ-series EtherNet/IP Units Operation Manual for NJ-series CPU Unit* (Cat. No. W495)

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- Execute the CIPOpen or CIPOpenWithDataSize instruction to obtain the value for *Handle* before you execute this instruction.
- You can use this instruction only through an NJ/NX-series CPU Unit, through a built-in EtherNet/IP port on an NY-series Controller, or through an EtherNet/IP Unit connected to an NJ-series CPU Unit.

- If a variable is read from an OMRON Controller, the variable must be published to the network. Publish the variable to the network in advance.
- You cannot specify an address in memory for CJ-series Units directly to read data. To read specific addresses in memory for CJ-series Units, use an AT specification in advance to assign the memory addresses to a variable.
- You cannot specify an address in local memory for CJ-series Units directly to store data. To store data in specific addresses in memory for CJ-series Units, use an AT specification in advance to assign the memory addresses to *DstDat*.
- The characters that can be used in *SrcDat* are specified in the following table.

Item	Specification
Maximum number of bytes	127 bytes
Character code	UTF-8
Applicable characters	Alphanumeric characters (not case sensitive), single-byte Katakana, multibyte characters, and '_' (underbars)
Prohibited text strings	• Any text string that starts with ASCII characters 0 to 9 (character codes 16#30 to 16#39) • A text string that consists of only a single _ (underbar) ASCII character • Any text string that includes two or more consecutive _ (underbar) ASCII characters • Any text string that starts with an _ (underbar) ASCII character • Any text string that ends with an _ (underbar) ASCII character • Any text string that starts with "P_"

- An error occurs in the following cases. *Error* will change to TRUE.
  - The value of *Size* is outside of the valid range.
  - The text string in *SrcDat* is not valid.
  - The data type of the value that was read does not agree with the data type of *DstDat*.
  - The size of data that was read exceeds the range of *DstDat*.
  - A data type that is not supported was specified for *DstDat.*
  - An error response defined by CIP was returned.
  - The value of *Handle.Handle* is outside of the valid range.
  - More than 32 CIP-related instructions were executed simultaneously.
  - The connection that was established with the CIPOpen or CIPOpenWithDataSize instruction has timed out.
  - The size of *SrcDat* exceeded the data size determined by the instruction that established the connection and the data type of the read data.

- For this instruction, expansion error code *ErrorIDEx* gives the CIP message error code. The meanings are as follows:

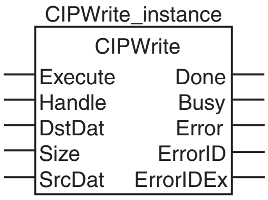
Value	Error
16#02000000	Normal communications are not possible due to a high load at the remote node.
16#04000000	The specified source variable is one of the following data types and it does not exist on the other Controller. • Basic data type • Enumeration • Structure • Union • Array
16#05000000	The specified source variable is one of the following and it does not exist on the other Controller. • Enumeration enumerator • Structure member • Union member • Array element
16#08000000	The requested service does not support.
16#0C008010	The specified source variable is being downloaded.
16#0C008011	
16#11000000	The value of Size is exceeds the data size that can currently be read.
16#1F000102	The variable to read is a variable that is not possible to read.
16#1F008007	The inaccessible variable is specified.
16#20008017	The specified source variable is not an array and the number of elements to read is not 1.
16#20008018	The specified source variable is an array and the number of elements to read exceeds the number of elements in the array.
16#26000000	The specified destination variable contains only the NULL character.

Sample Programming

Refer to the sample programming that is provided for the CIPOpen instruction (page 2-998).

CIPWrite

The CIPWrite instruction uses a class 3 explicit message to write the value of a variable in another Controller on a CIP network.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
CIPWrite	Write Variable Class 3 Explicit	FB		CIPWrite_instance(Execute, Handle, DstDat, Size, SrcDat, Done, Busy, Error, ErrorID, ErrorIDEx);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Handle	Handle	Input	Handle obtained with CIPOpen or CIPOpenWithDataSize instruction	---	---	---
DstDat	Destination variable name	Input	Name of variable to write in another Controller	Depends on data type.	---	''
Size	Number of elements to write	Input	Number of elements to write	0 to 8,178*1	---	1
SrcDat	Source data	Input	Data value to write	Depends on data type.	---	*2

*1 The range is 0 to 1,980 for NX1P2 and NJ-series CPU Units.

*2 If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Handle	Refer to *Function* for details on the structure _sCIP_HANDLE.																			
DstDat																				OK
Size							OK													
SrcDat	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
Handle	An enumeration, array*, structure, structure member, or union member can also be specified.																			

* You cannot specify a STRING array.

Function

The CIPWrite instruction writes the value of the network variable specified with destination variable name *DstDat* at another Controller on a CIP network. The other Controller is specified with *Handle*. The content of source data *ScrDat* is written.

Size specifies the number of elements to write. If *DstDat* is an array, specify the number of elements to write with *Size*. If *DstDat* is not an array, always specify 1 for *Size*. If the value of *Size* is 0, nothing is written regardless of whether *DstDat* is an array or not.

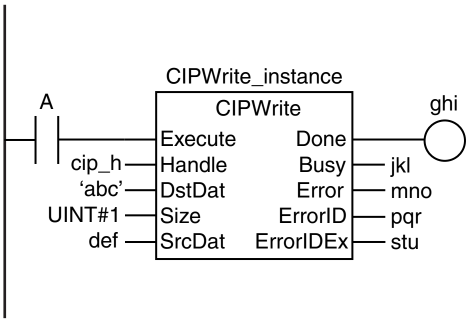
The data type of *Handle* is structure _sCIP_HANDLE. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Handle	Handle	Handle	_sCIP_ HANDLE	---	---	---
Handle	Handle	Handle	UDINT	Depends on data type.	---	---

If the value of *ErrorID* is WORD#16#1C00, the CIP message error code is stored in *ErrorIDEx*.

The following example writes the value of variable *def* from the local Controller to the variable *abc* in the remote Controller. The number of elements to write *Size* is UINT#1.

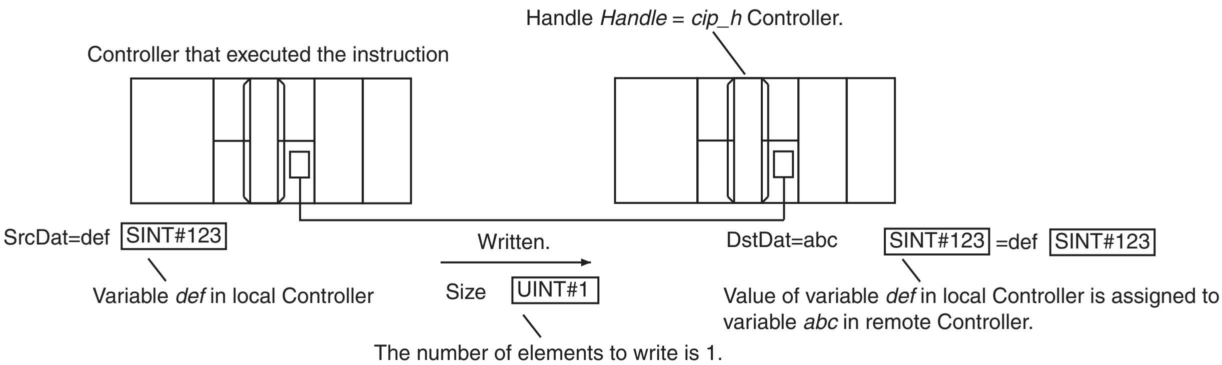
LD



ST

```
CIPWrite_instance(A, cip_h, 'abc', UINT#1, def,
        ghi, jkl, mno, pqr, stu);
```

The value of variable *SrcDat* in the local Controller is assigned to variable *DstDat* in the remote Controller on the CIP network specified by the handle *Handle*.
Size specifies the number of elements to write.

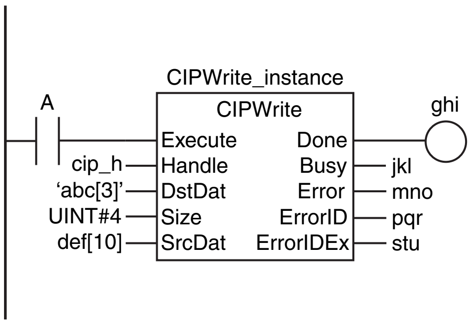


Writing Arrays

To write array data, pass a subscripted array element to *DstDat* as the parameter. Also pass a subscripted array element to *SrcDat* as the parameter.

The following example stores the contents of array variable elements *def[10]* to *def[13]* in the four array variable elements *abc[3]* to *abc[6]*.

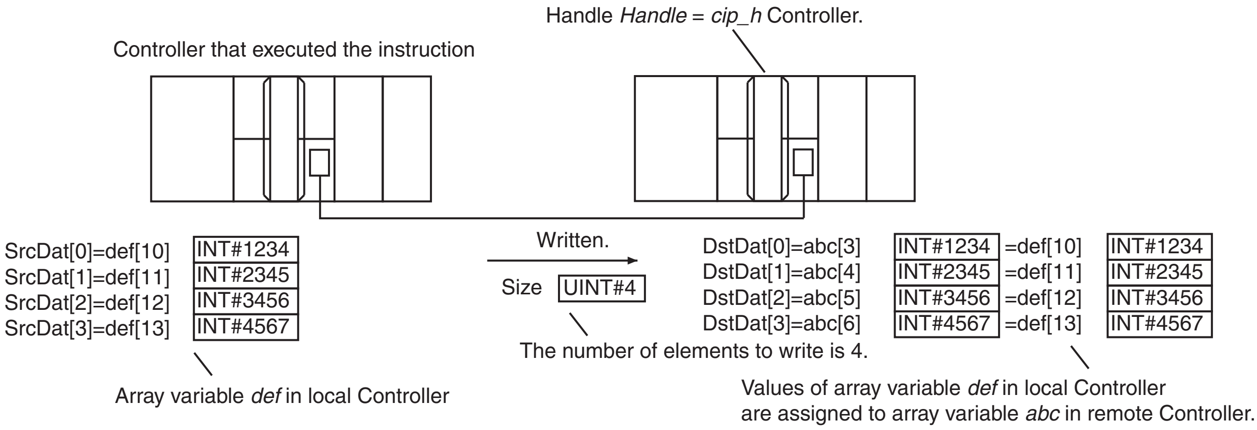
LD



ST

```
CIPWrite_instance(A, cip_h, 'abc[3]', UINT#4, def,
        ghi[10], jkl, mno, pqr, stu);
```

Values of array variable elements *def[10]* to *def[13]* in local Controller are assigned to array variable elements *abc[3]* Ito *abc[6]* in remote Controller.



Maximum Write Data Size

The maximum size of the data that you can write depends on the data type and variable name that are specified for *DstDat*, as given in the following table.

Maximum write data size [bytes] = Base size – Size of variable name of *DstDat*

Item in above formula	Meaning
Base size	Connections established with the CIPOpen instruction • Data type of variable specified for *DstDat* is a structure: 1,984 bytes • Data type of variable specified for *DstDat* is a STRING: 1,986 bytes • Other data types: 1,988 bytes
	Connections established with the CIPOpenWithDataSize instruction • Use the following formula when the data type of variable specified for *DstDat* is a structure: Base size [bytes] = *DataSize* in CIPOpenWithDataSize instruction – 10 • Use the following formula when the data type of variable specified for *DstDat* is a STRING: Base size [bytes] = *DataSize* in CIPOpenWithDataSize instruction – 8 • Use the following formula for other data types. Base size [bytes] = *DataSize* in CIPOpenWithDataSize instruction – 6

Item in above formula	Meaning
Size of variable name of *DstDat*	• The size of the variable name is calculated as the total bytes for the ASCII characters in all structure levels plus two times the number of levels. • If the number of bytes of ASCII characters in a level is an odd number, add 1. • If a level in the structure is an array, add four times the number of dimensions in the array. • Periods and commas in the structure and arrays are not included in the variable name size. Example 1: When the Variable Name of *DstDat* Is *aaa.bbbbb[1,2,3].cc* • The text string “aaa” in the first level is 3 bytes. It is an odd number, so 1 is added to make 4 bytes. • The text string “bbbbb[1,2,3]” in the second level is 5 bytes. It is an odd number, so 1 is added to make 6 bytes. • Also *bbbbb[1,2,3]* is a three-dimensional array, so 3 times 4, or 12, is added to make 18 bytes. • The text string “cc” in the third level is 2 bytes. It is an even number, so 2 bytes is used in the calculation. • If we add the number of levels 3 times 2, or 6, to 4 bytes for the first level, 18 bytes for the second level, and 2 bytes for the third level, the size of the variable name come to 30 bytes. Example 2: When the Variable Name of *DstDat* Is *val* • The text string “val” in the first level is 3 bytes. It is an odd number, so 1 is added to make 4 bytes. • If we then add the number of levels 1 times 2, or 2, the size of the variable name is 6 bytes. Example 3: When the Variable Name of *DstDat* Is *array[8].* • The text string “array” in the first level is 5 bytes. It is an odd number, so 1 is added to make 6 bytes. • It is a one-dimensional array. Therefore, 1 times 4, or 4, is added. • If we then add the number of levels 1 times 2, or 2, the size of the variable name is 12 bytes.

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta*1 _EIP1_EtnOnlineSta*2 _EIP2_EtnOnlineSta*3 _EIPIn1_EtnOnlineSta*4	Online	BOOL	This variable indicates when built-in EtherNet/IP port communications can be used. TRUE: Communications are possible. FALSE: Communications are not possible.

*1 Use this variable name for an NJ-series CPU Unit.

*2 Use this variable name for port 1 on an NX-series CPU Unit, or for an NY-series Controller.

*3 Use this variable name for port 2 on an NX-series CPU Unit.

*4 Use this variable name for the internal communication port on an NY-series Controller.

Additional Information

Refer to the following manuals for details on CIP communications.

- *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User’s Manual* (Cat. No. W506)
- *NY-series Industrial Panel PC / Industrial Box PC Built-in EtherNet/IP Port User’s Manual* (Cat. No. W563)
- *CJ-series EtherNet/IP Units Operation Manual for NJ-series CPU Unit* (Cat. No. W495)

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- Execute the CIPOpen or CIPOpenWithDataSize instruction to obtain the value for *Handle* before you execute this instruction.
- Always use a variable for the input parameter to pass to *SrcDat*. A building error will occur if a constant is passed.
- If *SrcDat* is an enumeration, you cannot directly pass an enumerator to it. A building error will occur if an enumerator is passed to it directly.
- You can use this instruction only through an NJ/NX-series CPU Unit, through a built-in EtherNet/IP port on an NY-series Controller, or through an EtherNet/IP Unit connected to an NJ-series CPU Unit.
- If a variable is written to an OMRON Controller, the variable must be published to the network. Publish the variable to the network in advance.
- You cannot specify an address in memory for CJ-series Units directly to write data. To write specific addresses in memory for CJ-series Units, use an AT specification in advance to assign the memory addresses to a variable.
- You cannot directly specify an address in local memory for CJ-series Units. To write specific addresses in memory for CJ-series Units, use an AT specification in advance to assign the memory addresses to *SrcDat*.
- The characters that can be used in *DstDat* are specified in the following table.

Item	Specification
Maximum number of bytes	127 bytes
Character code	UTF-8
Applicable characters	Alphanumeric characters (not case sensitive), single-byte Katakana, multibyte characters, and ‘_’(underbars)
Prohibited text strings	• Any text string that starts with ASCII characters 0 to 9 (character codes 16#30 to 16#39) • A text string that consists of only a single _ (underbar) ASCII character • Any text string that includes two or more consecutive _ (underbar) ASCII characters • Any text string that starts with an _ (underbar) ASCII character • Any text string that ends with an _ (underbar) ASCII character • Any text string that starts with “P_”

- An error occurs in the following cases. *Error* will change to TRUE.
  - The value of *Size* is outside of the valid range.
  - The text string in *DstDat* is not valid.
  - The value of *Size* exceeds the range of *SrcDat*.
  - A data type that is not supported was specified for *SrcDat.*
  - An error response defined by CIP was returned.
  - The value of *Handle.Handle* is outside of the valid range.
  - More than 32 CIP-related instructions were executed simultaneously.
  - The connection that was established with the CIPOpen or CIPOpenWithDataSize instruction has timed out.
  - The total of the size in *DstDat* and the value of *SrcDat* exceeded the data size determined by the instruction that established the connection and the data type of the write data.

• For this instruction, expansion error code *ErrorIDEx* gives the CIP message error code. The meanings are as follows:

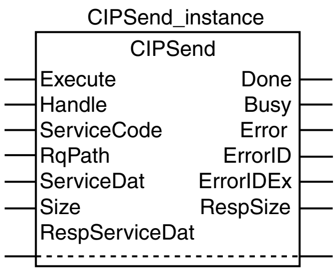
Value	Error
16#02000000	Normal communications are not possible due to a high load at the remote node.
16#04000000	The specified source variable is one of the following data types and it does not exist on the other Controller. • Basic data type • Enumeration • Structure • Union • Array
16#05000000	The specified source variable is one of the following and it does not exist on the other Controller. • Enumeration enumerator • Structure member • Union member • Array element
16#08000000	The requested service does not support.
16#0C008010	The specified source variable is being downloaded.
16#0C008011	
16#1F000102	• The specified destination variable has a Constant attribute, so it cannot be written. • The write data does not agree with the number of write elements.
16#1F008007	The inaccessible variable is specified.
16#20008017	The specified destination variable is not an array and the number of elements to write is not 1.
16#20008018	The specified destination variable is an array and the number of elements to write exceeds the number of elements in the array.
16#20008028	• The specified destination variable is an enumeration and the write data is not the value of an enumerator. • The specified destination variable has a Range Specification attribute and the write data is out of range.
16#26000000	The specified destination variable contains only the NULL character.

Sample Programming

Refer to the sample programming that is provided for the CIPOpen instruction (page 2-998).

CIPSend

The CIPSend instruction sends a class 3 CIP message to a specified device on a CIP network.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
CIPSend	Send Explicit Message Class 3	FB		CIPSend_instance(Execute, Handle, ServiceCode, RqPath, ServiceDat, Size, RespServiceDat, Done, Busy, Error, ErrorID, ErrorIDEx, RespSize);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Handle	Handle	Input	Handle obtained with CIPOpen or CIPOpenWithDataSize instruction	---	---	---
ServiceCode	Service code		Service code	Depends on data type.		
RqPath	Request path		Request path	---		
ServiceDat	Service data		Service data to send	Depends on data type.		
Size	Number of elements to send		Number of elements to send			*
RespServiceDat	Response data	In-out	Response data	Depends on data type.	---	---
RespSize	Response size	Output	Response data size	Depends on data type.	Bytes	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Handle	Refer to *Function* for details on the structure _sCIP_HANDLE.																			
ServiceCode		OK																		
RqPath	Refer to *Function* for details on the structure _sREQUEST_PATH or _sREQUEST_PATH_EX*1.																			
ServiceDat		OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
	An array, structure member, or union member can also be specified.																			
Size							OK													
RespService-Dat		OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
	An array, structure member, or union member can also be specified.																			
RespSize							OK													

*1 A CPU Unit with unit version 1.11 or later and Sysmac Studio version 1.15 or higher are required to specify _sREQUEST_PATH_EX type.

Function

The CIPSend instruction sends service data *ServiceDat* for the service specified with service code *ServiceCode* as a class 3 explicit message.
The destination is specified with handle *Handle*.
RqPath specifies the request path.

Size specifies the number of elements to send. If *ServiceDat* is an array, specify the number of elements to send with *Size*. If *ServiceDat* is not an array, always specify 1 for *Size*. If no service data is required, set *Size* to 0.
The response data received later is stored in *RespServiceDat*. The number of bytes of the response data is stored in *RespSize*.

The data type of *Handle* is structure _sCIP_HANDLE. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Handle	Handle	Handle	_sCIP_ HANDLE	---	---	---
Handle	Handle	Handle	UDINT	Depends on data type.	---	---

The data type of *RqPath* is structure _sREQUEST_PATH or _sREQUEST_PATH_EX.
Normally, use _sREQUEST_PATH. When you want to specify any logical format size, use _sREQUEST_PATH_EX. The specifications are as follows:

- _sREQUEST_PATH type

Name	Meaning	Description	Data type	Valid range	Unit	Default
RqPath	Request path	Request path	_sREQUEST_-PATH	---	---	---
ClassID	Class ID	Class ID	UINT	Depends on data type.	---	0
InstanceID	Instance ID	Instance ID	UINT			
isAttributeID	Attribute usage	TRUE:Attribute ID used. FALSE:Attribute ID not used.	BOOL			FALSE
AttributeID	Attribute ID	Attribute ID	UINT			0

Note The logical format size of each ID in _sREQUEST_PATH type is 16 bits.

- _sREQUEST_PATH_EX type

Name	Meaning	Description	Data type	Valid range	Unit	Default
RqPath	Request path	Request path	_sREQUEST_PATH_EX	---	---	---
ClassIDLogicalFormat	Class ID logical format	Class ID data size	_eCIP_LOGICAL_FORMAT	Depends on data type.	---	_8BIT
ClassID	Class ID	Class ID	UDINT			0
InstanceIDLogicalFormat	Instance ID logical format	Instance ID data size	_eCIP_LOGICAL_FORMAT			_8BIT
InstanceID	Instance ID	Instance ID	UDINT			0
isAttributeID	Attribute usage	TRUE:Attribute ID used. FALSE:Attribute ID not used.	BOOL			FALSE
AttributeIDLogicalFormat	Attribute ID logical format	Attribute ID data size	_eCIP_LOGICAL_FORMAT			_8BIT
AttributeID	Attribute ID	Attribute ID	UDINT			0

The data type of *ClassIDLogicalFormat*, *InstanceIDLogicalFormat*, and *AttributeIDLogicalFormat* is enumerated type _eCIP_LOGICAL_FORMAT.

The meanings of the enumerators of enumerated type _eCIP_LOGICAL_FORMAT are as follows:

Enumerator	Meaning
_8BIT	8 bits
_16BIT	16 bits
_32BIT	32 bits

If the value of *ErrorID* is WORD#16#1C00, the CIP message error code is stored in *ErrorIDEx*. The meaning and values of *ErrorIDEx* depend on the remote node. Refer to the manual for the remote node.

Sending and Receiving Arrays

If *ServiceDat* or *RespServiceDat* is an array, pass a subscripted array element to it as the parameter.

Maximum Read/Write Data Size

The maximum size of the data that you can read depends on whether the connection was opened with the CIPOpen instruction or the CIPOpenWithDataSize instruction as shown in the following table.

Instruction that opened the connection	Maximum size of data that you can read
CIPOpen	1,990 bytes
CIPOpenWithDataSize	With 8,188* as the upper limit, responses returned by the server can be read.

* The maximum size is 1,990 for NX1P2 and NJ-series CPU Units.

The maximum size of the data that you can write depends on whether there is a request path attribute and the instruction that established the connection, as given below.

Maximum write data size [bytes] = Base size – Attribute usage

Item in above formula	Meaning
Base size	• Connection established with the CIPOpen instruction: 1,992 bytes • Connection established with the CIPOpenWithDataSize instruction: *DataSize* in CIPOpenWithDataSize instruction – 2
Attribute usage*1	Attribute ID used: 14 bytes Attribute ID not used: 10 bytes

*1 With a CPU Unit with unit version 1.10 or earlier or Sysmac Studio version 1.14 or lower, the values are as follows:
Attribute ID used: 12 bytes
Attribute ID not used: 8 bytes

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta*1	Online	BOOL	This variable indicates when built-in EtherNet/IP port communications can be used. TRUE: Communications are possible. FALSE: Communications are not possible.
_EIP1_EtnOnlineSta*2 _EIP2_EtnOnlineSta*3			
_EIPIn1_EtnOnlineSta*4			

*1 Use this variable name for an NJ-series CPU Unit.

*2 Use this variable name for port 1 on an NX-series CPU Unit, or for an NY-series Controller.

*3 Use this variable name for port 2 on an NX-series CPU Unit.

*4 Use this variable name for the internal communication port on an NY-series Controller.

Additional Information

Refer to the following manuals for details on CIP communications.

- *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User's Manual* (Cat. No. W506)
- *NY-series Industrial Panel PC / Industrial Box PC Built-in EtherNet/IP Port User's Manual* (Cat. No. W563)
- *CJ-series EtherNet/IP Units Operation Manual for NJ-series CPU Unit* (Cat. No. W495)

Precautions for Correct Use

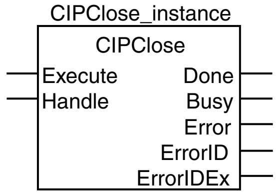
- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- Execute the CIPOpen or CIPOpenWithDataSize instruction to obtain the value for Handle before you execute this instruction.
- Always use a variable for the input parameter to pass to *ServiceDat*. A building error will occur if a constant is passed.
- You can use this instruction only through an NJ/NX-series CPU Unit, through a built-in EtherNet/IP port on an NY-series Controller, or through an EtherNet/IP Unit connected to an NJ-series CPU Unit.
- If a variable is written to an OMRON Controller, the variable must be published to the network. Publish the variable to the network in advance.
- An error occurs in the following cases. *Error* will change to TRUE.
  - A value that is out of valid range is set for *RqPath.ClassIDLogicalFormat* or *RqPath.AttributeIDLogicalFormat*.
  - A mismatch occurred between the following two variables: the size specified for *RqPath.ClassIDLogicalFormat* and the data size of *RqPath.ClassID*, the size specified for *RqPath.InstanceIDLogicalFormat* and the data size of *RqPath.InstanceID*, or the size specified for *RqPath.AttributeIDLogicalFormat* and the data size of *RqPath.AttributeID*.
  - The value of *Size* exceeds the write data range.
  - The value of *Size* exceeds the range of *ServiceDat*.
  - The value of *RespSize* exceeds the range of *RespServiceDat*.
  - A data type that is not supported was specified for *ServiceDat.*
  - A data type that is not supported was specified for *RespServiceDat.*
  - A variable whose data type is other than _sREQUEST_PATH or _sREQUEST_PATH_EX is specified for *RqPath*.
  - An error response defined by CIP was returned.
  - The value of *Handle.Handle* is outside of the valid range.
  - More than 32 CIP-related instructions were executed simultaneously.
  - The connection that was established with the CIPOpen or CIPOpenWithDataSize instruction has timed out.
  - The total of the sizes of *RqPath* and *ServiceDat* exceeded the data size determined by the instruction that established the connection.

Sample Programming

Refer to the sample programming that is provided for the CIPOpen instruction (page 2-998).

CIPClose

The CIPClose instruction closes the CIP class 3 connection to the specified handle.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
CIPClose	Close CIP Class 3 Connection	FB		CIPClose_instance(Execute, Handle, Done, Busy, Error, ErrorID, ErrorIDEx);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Handle	Handle	Input	Handle obtained with CIPOpen or CIPOpenWithDataSize instruction	---	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Handle	Refer to *Function* for details on the structure _sCIP_HANDLE.																			

Function

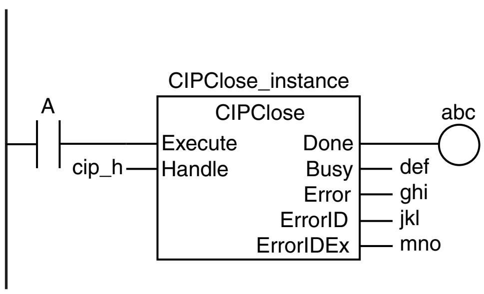
The CIPClose instruction closes the CIP class 3 connection specified with the handle *Handle*.

The data type of *Handle* is structure _sCIP_HANDLE. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Handle	Handle	Handle	_sCIP_HANDLE	---	---	---
Handle	Handle	Handle	UDINT	Depends on data type.	---	---

The following figure shows a programming example. The CIPClose instruction closes the CIP class 3 connection specified with *Handle* (= cip_h).

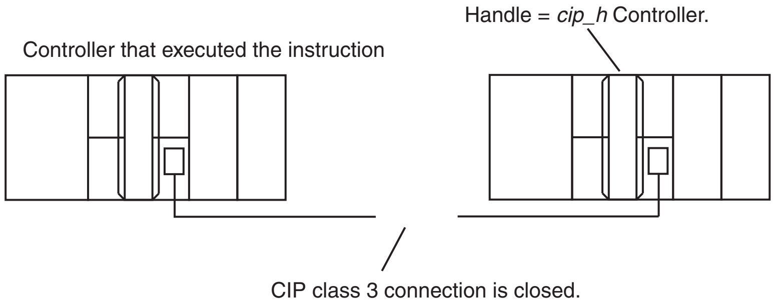
LD



ST

```
CIPClose_instance(A, cip_h, abc, def,
                  ghi, jkl, mno);
```

The CIPClose instruction closes the CIP class 3 connection specified with *Handle*.



Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta*1 _EIP1_EtnOnlineSta*2 _EIP2_EtnOnlineSta*3 _EIPIn1_EtnOnlineSta*4	Online	BOOL	This variable indicates when built-in EtherNet/IP port communications can be used. TRUE: Communications are possible. FALSE: Communications are not possible.

*1 Use this variable name for an NJ-series CPU Unit.

*2 Use this variable name for port 1 on an NX-series CPU Unit, or for an NY-series Controller.

*3 Use this variable name for port 2 on an NX-series CPU Unit.

*4 Use this variable name for the internal communication port on an NY-series Controller.

Additional Information

Refer to the following manuals for details on CIP communications.

- *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User's Manual* (Cat. No. W506)
- *NY-series Industrial Panel PC / Industrial Box PC Built-in EtherNet/IP Port User's Manual* (Cat. No. W563)
- *CJ-series EtherNet/IP Units Operation Manual for NJ-series CPU Unit* (Cat. No. W495)

Precautions for Correct Use

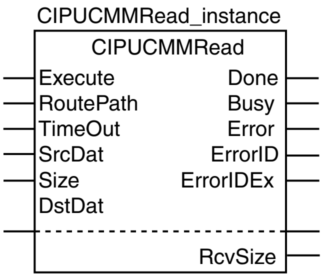
- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- Specify the handle that was obtained with the CIPOpen or CIPOpenWithDataSize instruction for *Handle*.
- You can use this instruction only through an NJ/NX-series CPU Unit, through a built-in EtherNet/IP port on an NY-series Controller, or through an EtherNet/IP Unit connected to an NJ-series CPU Unit.
- This instruction does not use *ErrorIDEx*.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The value of *Handle.Handle* is outside of the valid range.
  - More than 32 CIP-related instructions were executed simultaneously.

Sample Programming

Refer to the sample programming that is provided for the CIPOpen instruction (page 2-998).

CIPUCMMRead

The CIPUCMMRead instruction uses a UCMM explicit message to read the value of a variable in another Controller on the specified CIP network.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
CIPUCMM Read	Read Variable UCMM Explicit	FB		CIPUCMMRead_instance(Execute, RoutePath, TimeOut, SrcDat, Size, DstDat, Done, Busy, Error, ErrorID, ErrorIDEx, RcvSize);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
RoutePath	Route path	Input	Route path	Depends on data type.	–---	---
TimeOut	Timeout time		Timeout time	1 to 65535	0.1 s	20 (2 s)
SrcDat	Source variable name		Name of variable to read in other Controller	Depends on data type.	---	"
Size	Number of elements to read		Number of elements to read	0 to 496		1
DstDat	Read data	In-out	Read data value	Depends on data type.	---	---
RcvSize	Read data size	Output	Read data size	0 to 496	Bytes	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
RoutePath								OK												OK
TimeOut							OK													
SrcDat																				OK
Size							OK													
DstDat	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	An enumeration, array, structure, structure member, or union member can also be specified.*																			
RcvSize									OK											

* You cannot specify a STRING array.

Function

The CIPUCMMRead instruction reads the value of the network variable specified with source variable name *SrcDat* from another Controller on a CIP network. The other Controller is specified with route path *RoutePath*.
The read data value is stored in *DstDat.*

Size specifies the number of elements to read. If *SrcDat* is an array, specify the number of elements to read with *Size*. If *SrcDat* is not an array, always specify 1 for *Size*. If the value of *Size* is 0, nothing is read regardless of whether *SrcDat* is an array or not.
When the read operation is completed, the number of bytes of the data that was read is assigned to read data size *RcvSize*. The maximum size of the data that you can read depends on the data type of the variable as follows:

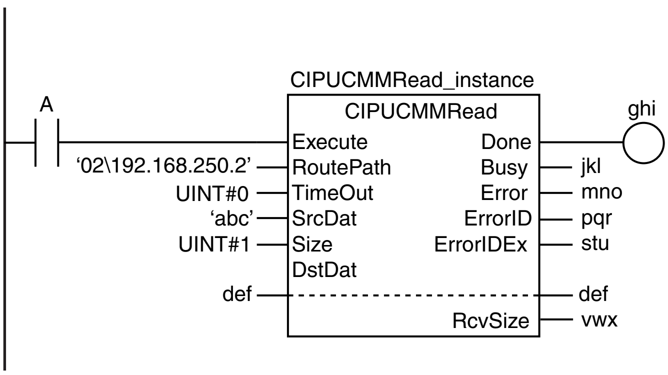
- Structure: 492 bytes
- STRING: 494 bytes
- Other data types: 496 bytes

TimeOut specifies the timeout time. If a response does not return within the timeout time, it is assumed that communications failed.

If the value of *ErrorID* is WORD#16#1C00, the CIP message error code is stored in *ErrorIDEx*.

In the following example, the value of variable *abc* in the remote Controller is read and stored in the variable *def* in the local Controller. The number of elements to read *Size* is UINT#1. The data type of *abc* and *def* is SINT. The size of SINT data is one byte, so the value of the read data size *vwx* is UINT#1.

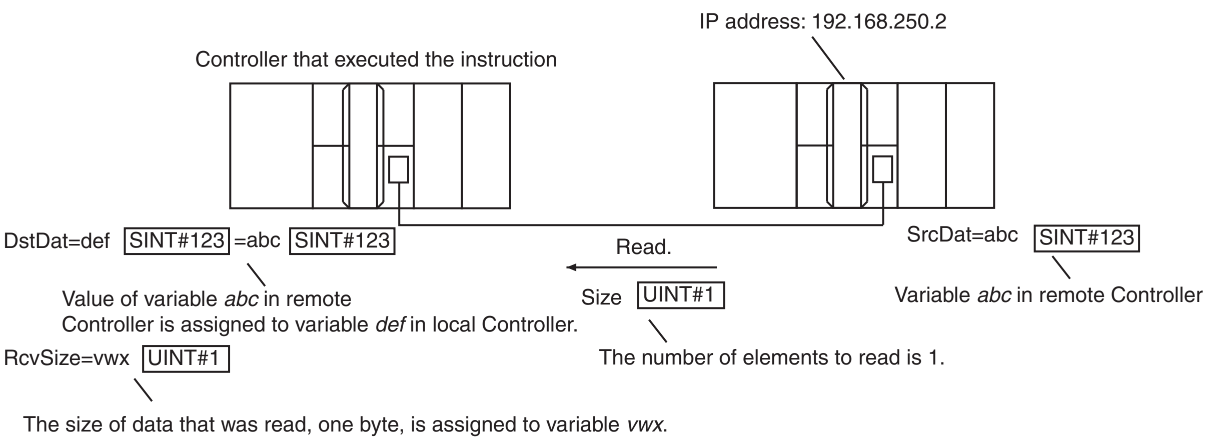
LD



ST

```
CIPUCMMRead_instance(A, '2\192.168.250.2',
    UINT#0, 'abc', UINT#1,
    def, ghi, jkl, mno, pqr,
    stu, vwx);
```

Value of variable *SrcDat* in remote Controller on the CIP network specified by the route path *RoutePath* is assigned to variable *DstDat* in local Controller. *Size* specifies the number of elements to read. The size of data that was read is assigned to *RcvSize*.

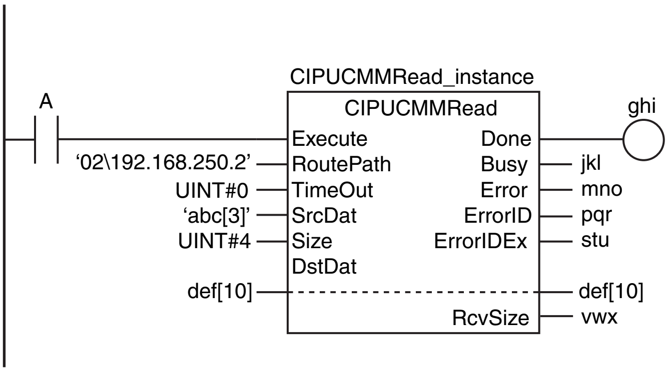


Reading Arrays

To read array data, pass a subscripted array element to *ScrDat* as the parameter. Also pass a subscripted array element to *DstDat* as the parameter.

The following example reads the four array variable elements *abc[3]* to *abc[6]* from the remote Controller and stores the results in array variable elements *def[10]* to *def[13]* in the local Controller. The data type of *abc* and *def* is INT. The size of INT data is two bytes, so the value of the read data size *vwx* is UINT#8.

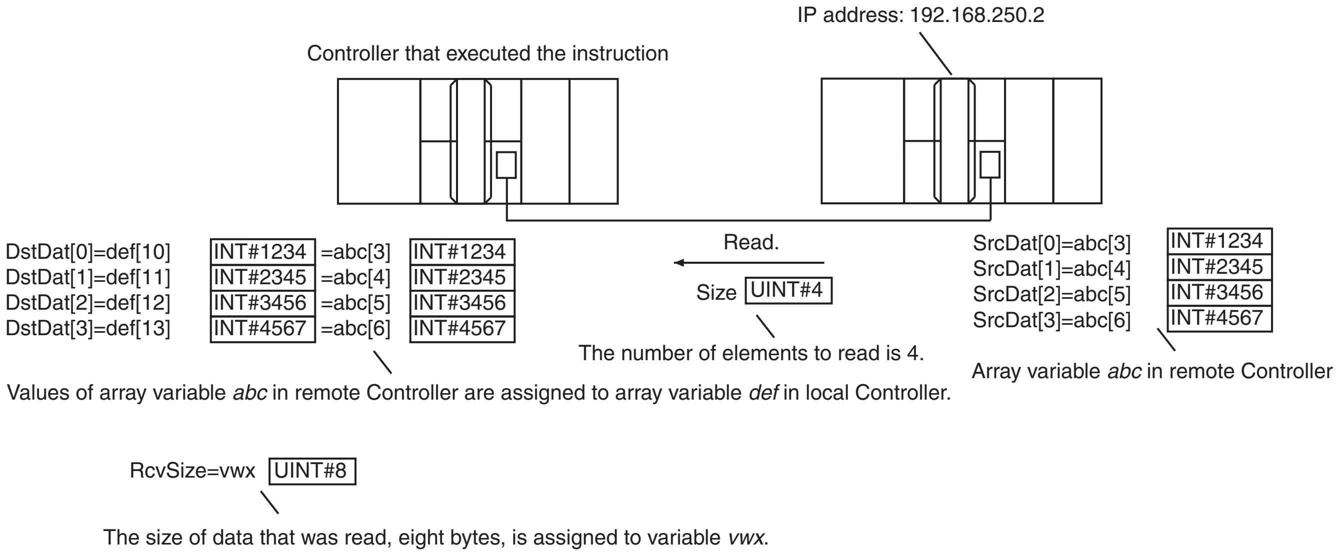
LD



ST

```
CIPUCMMRead_instance(A, '2\192.168.250.2',
                     UINT#0, 'abc[3]', UINT#4,
                     def[10], ghi, jkl, mno, pqr,
                     stu, vwx);
```

Values of array variable elements *abc[3]* to *abc[6]* in remote Controller are assigned to array variable elements *def[10]* to *def[13]* in local Controller.



Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta*1 _EIP1_EtnOnlineSta*2 _EIP2_EtnOnlineSta*3 _EIPIn1_EtnOnlineSta*4	Online	BOOL	This variable indicates when built-in EtherNet/IP port communications can be used. TRUE: Communications are possible. FALSE: Communications are not possible.

*1 Use this variable name for an NJ-series CPU Unit.

*2 Use this variable name for port 1 on an NX-series CPU Unit, or for an NY-series Controller.

*3 Use this variable name for port 2 on an NX-series CPU Unit.

*4 Use this variable name for the internal communication port on an NY-series Controller.

Additional Information

Refer to the following manuals for details on CIP communications.

- *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User's Manual* (Cat. No. W506)
- *NY-series Industrial Panel PC / Industrial Box PC Built-in EtherNet/IP Port User's Manual* (Cat. No. W563)
- *CJ-series EtherNet/IP Units Operation Manual for NJ-series CPU Unit* (Cat. No. W495)

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the built-in EtherNet/IP ports on NJ/NX-series CPU Units and NY-series Controllers.
- If a variable is read from an OMRON Controller, the variable must be published to the network. Publish the variable to the network in advance.
- You cannot specify an address in memory for CJ-series Units directly to read data. To read specific addresses in memory for CJ-series Units, use an AT specification in advance to assign the memory addresses to a variable.
- You cannot specify an address in local memory for CJ-series Units directly to store data. To store data in specific addresses in memory for CJ-series Units, use an AT specification in advance to assign the memory addresses to *DstDat*.
- The characters that can be used in *SrcDat* are specified in the following table.

Item	Specification
Maximum number of bytes	127 bytes
Character code	UTF-8
Applicable characters	Alphanumeric characters (not case sensitive), single-byte Katakana, multibyte characters, and '_' (underbars)
Prohibited text strings	• Any text string that starts with ASCII characters 0 to 9 (character codes 16#30 to 16#39) • A text string that consists of only a single _ (underbar) ASCII character • Any text string that includes two or more consecutive _ (underbar) ASCII characters • Any text string that starts with an _ (underbar) ASCII character • Any text string that ends with an _ (underbar) ASCII character • Any text string that starts with "P_"

- An error occurs in the following cases. *Error* will change to TRUE.
  - The value of *TimeOut* is outside of the valid range.
  - The value of *Size* is outside of the valid range.
  - The text string in *SrcDat* is not valid.
  - The data type of the value that was read does not agree with the data type of *DstDat*.
  - The size of data that was read exceeds the range of *DstDat*.
  - A data type that is not supported was specified for *DstDat.*
  - An error response defined by CIP was returned.
  - The text string in *RoutePath* is not valid.
  - More than 32 CIP-related instructions were executed simultaneously.
  - A response was not received even though the timeout time was exceeded.

  - There is a setting error for the local IP address.
  - The instruction was executed when there was a BOOTP server error.
  - A duplicated IP error occurred.
- For this instruction, expansion error code *ErrorIDEx* gives the CIP message error code. The meanings are as follows:

Value	Error
16#02000000	Normal communications are not possible due to a high load at the remote node.
16#04000000	The specified source variable is one of the following data types and it does not exist on the other Controller. • Basic data type • Enumeration • Structure • Union • Array
16#05000000	The specified source variable is one of the following and it does not exist on the other Controller. • Enumeration enumerator • Structure member • Union member • Array element
16#08000000	The requested service does not support.
16#0C008010	The specified source variable is being downloaded.
16#0C008011	
16#11000000	The value of Size is exceeds the data size that can currently be read.
16#1F000102	The variable to read is a variable that is not possible to read.
16#1F008007	The inaccessible variable is specified.
16#20008017	The specified source variable is not an array and the number of elements to read is not 1.
16#20008018	The specified source variable is an array and the number of elements to read exceeds the number of elements in the array.
16#26000000	The specified destination variable contains only the NULL character.

Sample Programming

Refer to the sample programming that is provided for the CIPUCMMSend instruction (page 2-1043).

CIPUCMMWrite

The CIPUCMMWrite instruction uses a UCMM explicit message to write the value of a variable in another Controller on a CIP network.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
CIPUCMM Write	Write Variable UCMM Explicit	FB	CIPUCMMWrite_instance CIPUCMMWrite Execute — Done RoutePath — Busy TimeOut — Error DstDat — ErrorID Size — ErrorIDEx SrcDat	CIPUCMMWrite_instance(Execute, RoutePath, TimeOut, DstDat, Size, SrcDat, Done, Busy, Error, ErrorID, ErrorIDEx);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
RoutePath	Route path	Input	Route path	Depends on data type.	---	---
TimeOut	Timeout time		Timeout time	1 to 65535	0.1 s	20 (2 s)
DstDat	Destination variable name		Name of variable to write in another Controller	Depends on data type.	---	''
Size	Number of elements to write		Number of elements to write	0 to 488		1
SrcDat	Source data		Data value to write	Depends on data type.		*

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
RoutePath																				OK
TimeOut							OK													
DstDat																				OK
Size							OK													
SrcDat	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	An enumeration, array*, structure, structure member, or union member can also be specified.																			

* You cannot specify a STRING array.

Function

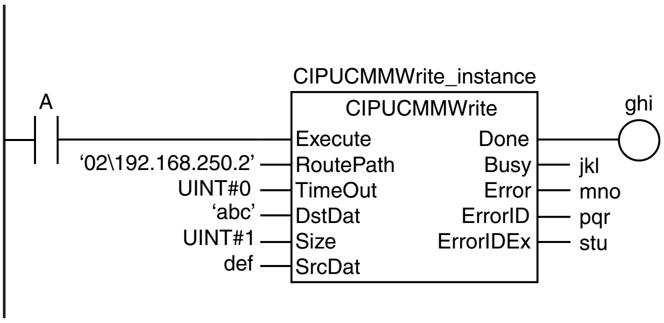
The CIPUCMMWrite instruction writes the value of the network variable specified with destination variable name *DstDat* at another Controller on a CIP network. The other Controller is specified with route path *RoutePath*.
The content of source data *ScrDat* is written.
Size specifies the number of elements to write. If *DstDat* is an array, specify the number of elements to write with *Size*. If *DstDat* is not an array, always specify 1 for *Size*. If the value of *Size* is 0, nothing is written regardless of whether *DstDat* is an array or not.

TimeOut specifies the timeout time. If a response does not return within the timeout time, it is assumed that communications failed.

If the value of *ErrorID* is WORD#16#1C00, the CIP message error code is stored in *ErrorIDEx*.

The following example writes the value of variable *def* from the local Controller to the variable *abc* in the remote Controller. The number of elements to write *Size* is UINT#1.

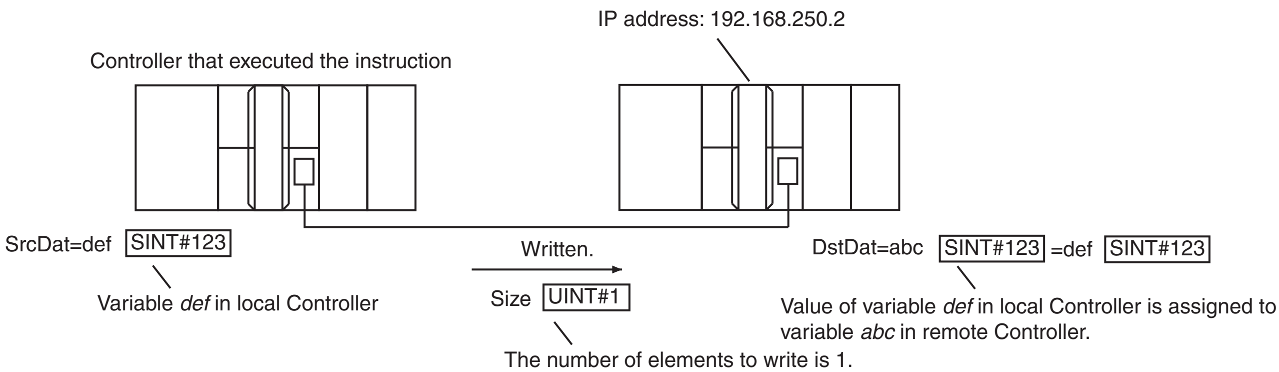
LD



ST

```
CIPUCMMWrite_instance(A, '2\192.168.250.2',
                      UINT#0, 'abc', UINT#1,
                      def, ghi, jkl, mno, pqr,
                      stu);
```

Value of variable *ScrDat* in local Controller is assigned to variable *DstDat* in remote Controller on the CIP network specified by the route path *RoutePath*. *Size* specifies the number of elements to write.

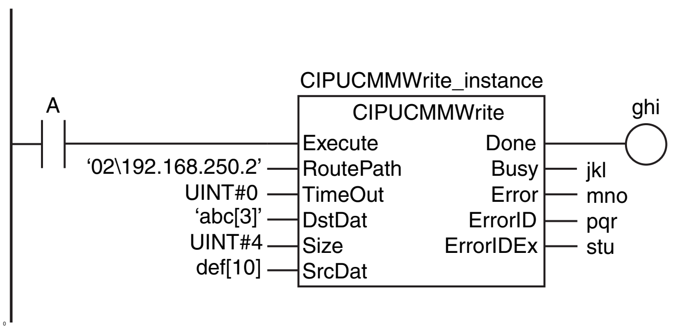


Writing Arrays

To write array data, pass a subscripted array element to *DstDat* as the parameter. Also pass a subscripted array element to *SrcDat* as the parameter.

The following example stores the contents of array variable elements *def[10]* to *def[13]* in the four array variable elements *abc[3]* to *abc[6]*.

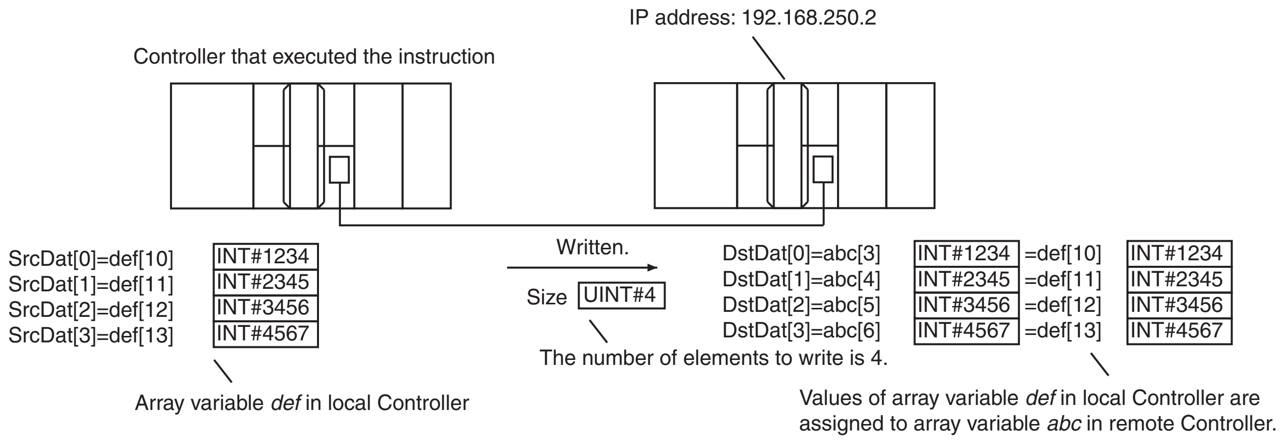
LD



ST

```
CIPUCMMWrite_instance(A, '2\192.168.250.2',
                      UINT#0, 'abc[3]', UINT#4,
                      def[10], ghi, jkl, mno, pqr,
                      stu);
```

Values of array variable elements *def[10]* to *def[13]* in local Controller are assigned to array variable elements *abc[3]* Ito *abc[6]* in remote Controller.



Maximum Write Data Size

The maximum size of the data that you can write depends on the data type and variable name that are specified for *DstDat* and the route path, as given in the following table.

Maximum write data size [bytes] = Base size – Size of variable name of *DstDat* – Path information size

Item in above formula	Meaning
Base size	• Data type of variable specified for *DstDat* is a structure: 492 bytes • Data type of variable specified for *DstDat* is a STRING: 494 bytes • Other data types: 496 bytes

Item in above formula	Meaning
Size of variable name of *DstDat*	• The size of the variable name is calculated as the total bytes for the ASCII characters in all structure levels plus two times the number of levels. • If the number of bytes of ASCII characters in a level is an odd number, add 1. • If a level in the structure is an array, add four times the number of dimensions in the array. • Periods and commas in the structure and arrays are not included in the variable name size. Example 1: When the Variable Name of *DstDat* Is *aaa.bbbbb[1,2,3].cc* • The text string "aaa" in the first level is 3 bytes. It is an odd number, so 1 is added to make 4 bytes. • The text string "bbbbb[1,2,3]" in the second level is 5 bytes. It is an odd number, so 1 is added to make 6 bytes. • Also *bbbbb[1,2,3]* is a three-dimensional array, so 3 times 4, or 12, is added to make 18 bytes. • The text string "cc" in the third level is 2 bytes. It is an even number, so 2 bytes is used in the calculation. • If we add the number of levels 3 times 2, or 6, to 4 bytes for the first level, 18 bytes for the second level, and 2 bytes for the third level, the size of the variable name come to 30 bytes. Example 2: When the Variable Name of *DstDat* Is val • The text string "val" in the first level is 3 bytes. It is an odd number, so 1 is added to make 4 bytes. • If we then add the number of levels 1 times 2, or 2, the size of the variable name is 6 bytes. Example 3: When the Variable Name of *DstDat* Is *array[8]*. • The text string "array" in the first level is 5 bytes. It is an odd number, so 1 is added to make 6 bytes. • It is a one-dimensional array. Therefore, 1 times 4, or 4, is added. • If we then add the number of levels 1 times 2, or 2, the size of the variable name is 12 bytes.

Item in above formula	Meaning

Path information size	• If there are no hops, the path information size is 0 bytes.*
	• If there are hops, the path information size is the route path size plus 12 bytes.
	• The route path size is the bytes size of the ASCII characters in the route path.
	• However, the following precautions apply.
	  • If the address portion starts with “#”, calculate the network and address portions as a total of 2 bytes.
	  • If the address portion does not start with “#”, calculate the network portion as 2 bytes.
	  • If the address portion does not start with “#” and the number of bytes in the ASCII characters for the address portion is an odd number, add 1 byte.
	  • Do not include the level separator, “\”, between levels of the route path in the route path size.
	  • Do not include the first hop in the route path size.
	Example 1: When the Route Path Is 01\#11\02\192.168.250.2\01\#01
	  • The first hop in the route path size is not included, so ignore ‘01\#11’ at the start of the path.
	  • The network type is ‘02’, so use 2 bytes in the calculation.
	  • The address portion is ‘192.168.250.2’, so use 13 bytes in the calculation. It is an odd number, so 1 is added to make 14 bytes.
	  • For the following ‘01\#01’, the address portion starts with “#”, so the network and address portions are calculated as a total of 2 bytes.
	  • If you add all of the above sizes, the size of the route path is 18 bytes.
	  • If we then add 12 bytes to the route path size, the path information size is 30 bytes.
	Example 2: When the Route Path Is 02\192.168.250.2\01\#00
	  • The first hop in the route path size is not included, so ignore ‘02\192.168.250.2’ at the start of the path.
	  • For the following ‘01#01’, the address portion starts with “#”, so the network and address portions are calculated as a total of 2 bytes.
	  • Therefore, the size of the route path is 2 bytes.
	  • If we then add 12 bytes to the route path size, the path information size is 14 bytes.
	Example 3: When the Route Path Is 02\192.168.250.2
	  • If there are no hops, the path information size is 0 bytes.

* A hop is routing between the sending node and receiving node. For example, if the route path is 02\192.168.250.2\01\#00, the message is first routed to the node with an IP address of 192.168.250.2 to send the message to unit address 00. This involves one hop.

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta*1 _EIP1_EtnOnlineSta*2 _EIP2_EtnOnlineSta*3 _EIPIn1_EtnOnlineSta*4	Online	BOOL	This variable indicates when built-in EtherNet/IP port communications can be used. TRUE: Communications are possible. FALSE: Communications are not possible.

*1 Use this variable name for an NJ-series CPU Unit.

*2 Use this variable name for port 1 on an NX-series CPU Unit, or for an NY-series Controller.

*3 Use this variable name for port 2 on an NX-series CPU Unit.

*4 Use this variable name for the internal communication port on an NY-series Controller.

Additional Information

Refer to the following manuals for details on CIP communications.

- *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User's Manual* (Cat. No. W506)
- *NY-series Industrial Panel PC / Industrial Box PC Built-in EtherNet/IP Port User's Manual* (Cat. No. W563)
- *CJ-series EtherNet/IP Units Operation Manual for NJ-series CPU Unit* (Cat. No. W495)

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- Always use a variable for the input parameter to pass to *SrcDat*. A building error will occur if a constant is passed.
- If *SrcDat* is an enumeration, you cannot directly pass an enumerator to it. A building error will occur if an enumerator is passed to it directly.
- You can use this instruction only through an NJ/NX-series CPU Unit, through a built-in EtherNet/IP port on an NY-series Controller, or through an EtherNet/IP Unit connected to an NJ-series CPU Unit.
- If a variable is written to an OMRON Controller, the variable must be published to the network. Publish the variable to the network in advance.
- You cannot specify an address in memory for CJ-series Units directly to write data. To write specific addresses in memory for CJ-series Units, use an AT specification in advance to assign the memory addresses to a variable.
- You cannot directly specify an address in local memory for CJ-series Units. To write specific addresses in memory for CJ-series Units, use an AT specification in advance to assign the memory addresses to *SrcDat*.
- The characters that can be used in *DstDat* are specified in the following table.

Item	Specification
Maximum number of bytes	127 bytes
Character code	UTF-8
Applicable characters	Alphanumeric characters (not case sensitive), single-byte Katakana, multibyte characters, and '_' (underbars)
Prohibited text strings	• Any text string that starts with ASCII characters 0 to 9 (character codes 16#30 to 16#39) • A text string that consists of only a single _ (underbar) ASCII character • Any text string that includes two or more consecutive _ (underbar) ASCII characters • Any text string that starts with an _ (underbar) ASCII character • Any text string that ends with an _ (underbar) ASCII character • Any text string that starts with "P_"

- An error occurs in the following cases. *Error* will change to TRUE.
  - The value of *TimeOut* is outside of the valid range.
  - The value of *Size* is outside of the valid range.
  - The text string in *DstDat* is not valid.
  - The value of *Size* exceeds the range of *SrcDat*.
  - A data type that is not supported was specified for *SrcDat.*

  - An error response defined by CIP was returned.
  - The text string in *RoutePath* is not valid.
  - More than 32 CIP-related instructions were executed simultaneously.
  - A response was not received even though the timeout time was exceeded.
  - There is a setting error for the local IP address.
  - A duplicated IP error occurred.
- For this instruction, expansion error code *ErrorIDEx* gives the CIP message error code. The meanings are as follows:

Value	Error
16#02000000	Normal communications are not possible due to a high load at the remote node.
16#04000000	The specified source variable is one of the following data types and it does not exist on the other Controller. • Basic data type • Enumeration • Structure • Union • Array
16#05000000	The specified source variable is one of the following and it does not exist on the other Controller. • Enumeration enumerator • Structure member • Union member • Array element
16#08000000	The requested service does not support.
16#0C008010	The specified destination variable is being downloaded.
16#0C008011	
16#1F000102	The specified destination variable has a Constant attribute, so it cannot be written.
16#1F008007	The inaccessible variable is specified.
16#20008017	The specified destination variable is not an array and the number of elements to write is not 1.
16#20008018	The specified destination variable is an array and the number of elements to write exceeds the number of elements in the array.
16#20008028	• The specified destination variable is an enumeration and the write data is not the value of an enumerator. • The specified destination variable has a Range Specification attribute and the write data is out of range.
16#26000000	The specified destination variable name is only the NULL character.

Sample Programming

Refer to the sample programming that is provided for the CIPUCMMSend instruction (page 2-1043).

CIPUCMMSend

The CIPUCMMSend instruction sends a UCMM CIP message to a specified device on a CIP network.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
CIPUCMM Send	Send Explicit Message UCMM	FB	CIPUCMMSend_instance CIPUCMMSend Execute — Done RoutePath — Busy TimeOut — Error ServiceCode — ErrorID RqPath — ErrorIDEx ServiceDat — RespSize Size RespServiceDat	CIPUCMMSend_instance(Execute, RoutePath, TimeOut, ServiceCode, RqPath, ServiceDat, Size, RespServiceDat, Done, Busy, Error, ErrorID, ErrorIDEx, RespSize);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
RoutePath	Route path	Input	Route path	Depends on data type.	---	---
TimeOut	Timeout time		Timeout time	1 to 65535	0.1 s	20 (2.0 s)
ServiceCode	Service code		Service code	Depends on data type.	---	---
RqPath	Request path		Request path	---		
ServiceDat	Command data		Data to send	Depends on data type.		*
Size	Number of elements to send		Number of elements to send			1
RespServiceDat	Response data	In-out	Response data	Depends on data type.	---	---
RespSize	Response size	Output	Response data size	Depends on data type.	Bytes	---

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
RoutePath																				OK
TimeOut							OK													
Service Code		OK																		
ReqPath	Refer to *Function* for details on the structure _sREQUEST_PATH or _sREQUEST_PATH_EX *1.																			
ServiceDat		OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
	An array, structure member, or union member can also be specified.																			
Size							OK													

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
RespServiceDat		OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK					
	An array, structure member, or union member can also be specified.																			
RespSize							OK													

*1 A CPU Unit with unit version 1.11 or later and Sysmac Studio version 1.15 or higher are required to specify _sREQUEST_PATH_EX type.

Function

The CIPUCMMSend instruction sends command data *ServiceDat* for the service specified with service code *ServiceCode* as a UCMM explicit message.
The destination is specified with route path *RoutePath*.
RqPath specifies the request path.

Size specifies the number of elements to send. If *ServiceDat* is an array, specify the number of elements to send with *Size*. If *ServiceDat* is not an array, always specify 1 for *Size*. If no service data is required, set *Size* to 0.

The response data received later is stored in *RespServiceDat*. The number of bytes of the response data is stored in *RespSize*.

TimeOut specifies the timeout time. If a response does not return within the timeout time, it is assumed that communications failed.

The data type of *RqPath* is structure _sREQUEST_PATH or _sREQUEST_PATH_EX.

Normally, use _sREQUEST_PATH. When you want to specify the logical format size, use _sREQUEST_PATH_EX. The specifications are as follows:

- _sREQUEST_PATH type

Name	Meaning	Description	Data type	Valid range	Unit	Default
RqPath	Request path	Request path	_sREQUEST_PATH	---	---	---
ClassID	Class ID	Class ID	UINT	Depends on data type.	---	0
InstanceID	Instance ID	Instance ID	UINT			
isAttributeID	Attribute usage	TRUE:Attribute ID used. FALSE:Attribute ID not used.	BOOL			FALSE
AttributeID	Attribute ID	Attribute ID	UINT			0

Note The logical format size of each ID in _sREQUEST_PATH type is 16 bits.

- _sREQUEST_PATH_EX type

Name	Meaning	Description	Data type	Valid range	Unit	Default
RqPath	Request path	Request path	_sREQUEST_PATH_EX	---	---	---
ClassIDLogicalFormat	Class ID logical format	Class ID data size	_eCIP_LOGICAL_FORMAT	Depends on data type.	---	_8BIT
ClassID	Class ID	Class ID	UDINT			0
InstanceIDLogicalFormat	Instance ID logical format	Instance ID data size	_eCIP_LOGICAL_FORMAT			_8BIT
InstanceID	Instance ID	Instance ID	UDINT			0
isAttributeID	Attribute usage	TRUE:Attribute ID used. FALSE:Attribute ID not used.	BOOL			FALSE
AttributeIDLogicalFormat	Attribute ID logical format	Attribute ID data size	_eCIP_LOGICAL_FORMAT			_8BIT
AttributeID	Attribute ID	Attribute ID	UDINT			0

The data type of *ClassIDLogicalFormat*, *InstanceIDLogicalFormat*, and *AttributeIDLogicalFormat* is enumerated type _eCIP_LOGICAL_FORMAT.

The meanings of the enumerators of enumerated type _eCIP_LOGICAL_FORMAT are as follows:

Enumerator	Meaning
_8BIT	8 bits
_16BIT	16 bits
_32BIT	32 bits

If the value of *ErrorID* is WORD#16#1C00, the CIP message error code is stored in *ErrorIDEx*. The meaning and values of *ErrorIDEx* depend on the remote node. Refer to the manual for the remote node.

Sending and Receiving Arrays

If *ServiceDat* or *RespServiceDat* is an array, pass a subscripted array element to it as the parameter.

Maximum Read/Write Data Size

You can read a maximum of 492 bytes of data. The maximum size of the data that you can write depends on whether there is a request path attribute and the route path that is used, as given below.

Maximum write data size [bytes] = Base size – Attribute usage – Path information size

Item in above formula	Meaning
Base size	500 bytes
Attribute usage*1	Attribute ID used: 14 bytes Attribute ID not used: 10 bytes

Item in above formula	Meaning

Path information size	• If there are no hops, the path information size is 0 bytes.[*2]
	• If there are hops, the path information size is the route path size plus 12 bytes.
	• The route path size is the bytes size of the ASCII characters in the route path.
	• However, the following precautions apply.
	  • If the address portion starts with “#”, calculate the network and address portions as a total of 2 bytes.
	  • If the address portion does not start with “#”, calculate the network portion as 2 bytes.
	  • If the address portion does not start with “#” and the number of bytes in the ASCII characters for the address portion is an odd number, add 1 byte.
	  • Do not include the level separator, “\”, between levels of the route path in the route path size.
	  • Do not include the first hop in the route path size.
	Example 1: When the Route Path Is 01\#11\02\192.168.250.2\01\#01
	  • The first hop in the route path size is not included, so ignore ‘01\#11’ at the start of the path.
	  • The network type is ‘02’, so use 2 bytes in the calculation.
	  • The address portion is ‘192.168.250.2’, so use 13 bytes in the calculation. It is an odd number, so 1 is added to make 14 bytes.
	  • For the following ‘01\#01’, the address portion starts with “#”, so the network and address portions are calculated as a total of 2 bytes.
	  • If you add all of the above sizes, the size of the route path is 18 bytes.
	  • If we then add 12 bytes to the route path size, the path information size is 30 bytes.
	Example 2: When the Route Path Is 02\192.168.250.2\01\#00
	  • The first hop in the route path size is not included, so ignore ‘02\192.168.250.2’ at the start of the path.
	  • For the following ‘01#01’, the address portion starts with “#”, so the network and address portions are calculated as a total of 2 bytes.
	  • Therefore, the size of the route path is 2 bytes.
	  • If we then add 12 bytes to the route path size, the path information size is 14 bytes.
	Example 3: When the Route Path Is 02\192.168.250.2
	  • If there are no hops, the path information size is 0 bytes.

*1 With a CPU Unit with unit version 1.10 or earlier or Sysmac Studio version 1.14 or lower, the values are as follows:
Attribute ID used: 12 bytes
Attribute ID not used: 8 bytes

*2 A hop is routing between the sending node and receiving node. For example, if the route path is 02\192.168.250.2\01\#00, the message is first routed to the node with an IP address of 192.168.250.2 to send the message to unit address 00. This involves one hop.

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta*1	Online	BOOL	This variable indicates when built-in EtherNet/IP port communications can be used. TRUE: Communications are possible. FALSE: Communications are not possible.
_EIP1_EtnOnlineSta*2 _EIP2_EtnOnlineSta*3			
_EIPIn1_EtnOnlineSta*4			

*1 Use this variable name for an NJ-series CPU Unit.

*2 Use this variable name for port 1 on an NX-series CPU Unit, or for an NY-series Controller.

*3 Use this variable name for port 2 on an NX-series CPU Unit.

*4 Use this variable name for the internal communication port on an NY-series Controller.

Additional Information

Refer to the following manuals for details on CIP communications.

- *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User's Manual* (Cat. No. W506)
- *NY-series Industrial Panel PC / Industrial Box PC Built-in EtherNet/IP Port User's Manual* (Cat. No. W563)
- *CJ-series EtherNet/IP Units Operation Manual for NJ-series CPU Unit* (Cat. No. W495)

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- Always use a variable for the input parameter to pass to *ServiceDat*. A building error will occur if a constant is passed.
- You can use this instruction only through an NJ/NX-series CPU Unit, through a built-in EtherNet/IP port on an NY-series Controller, or through an EtherNet/IP Unit connected to an NJ-series CPU Unit.
- If a variable is written to an OMRON Controller, the variable must be published to the network. Publish the variable to the network in advance.
- An error occurs in the following cases. *Error* will change to TRUE.
  - A value that is out of valid range is set for *RqPath.ClassIDLogicalFormat* or *RqPath.AttributeIDLogicalFormat*.
  - A mismatch occurred between the following two variables: the size specified for *RqPath.ClassIDLogicalFormat* and the data size of *RqPath.ClassID*, the size specified for *RqPath.InstanceIDLogicalFormat* and the data size of *RqPath.InstanceID*, or the size specified for *RqPath.AttributeIDLogicalFormat* and the data size of *RqPath.AttributeID*.
  - The value of *TimeOut* is outside of the valid range.
  - The value of *Size* exceeds the write data range.
  - The value of *Size* exceeds the range of *ServiceDat*.
  - The value of *RespSize* exceeds the range of *RespServiceDat*.
  - A data type that is not supported was specified for *ServiceDat.*
  - A data type that is not supported was specified for *RespServiceDat.*
  - A variable whose data type is other than _sREQUEST_PATH or _sREQUEST_PATH_EX is specified for *RqPath*.
  - There is a setting error for the local IP address.

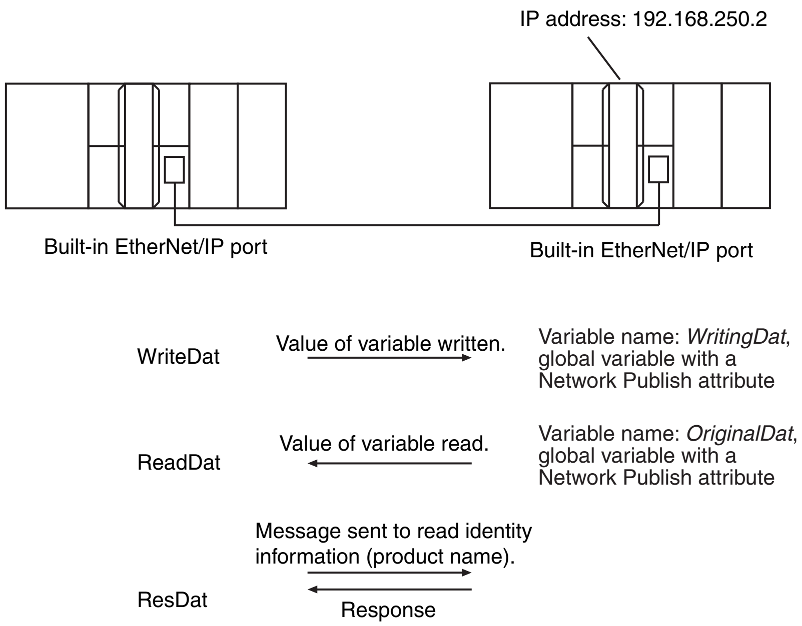
- A duplicated IP error occurred.
- The instruction was executed when there was a BOOTP server error.
- An error response defined by CIP was returned.
- The text string in *RoutePath* is not valid.
- More than 32 CIP-related instructions were executed simultaneously.
- A response was not received even though the timeout time was exceeded.

Sample Programming

This sample uses CIP UCMM messages to write a variable, read a variable, and send a message. The Controllers are connected to an EtherNet/IP network. The IP address of the remote node is 192.168.250.2.
The following procedure is used.

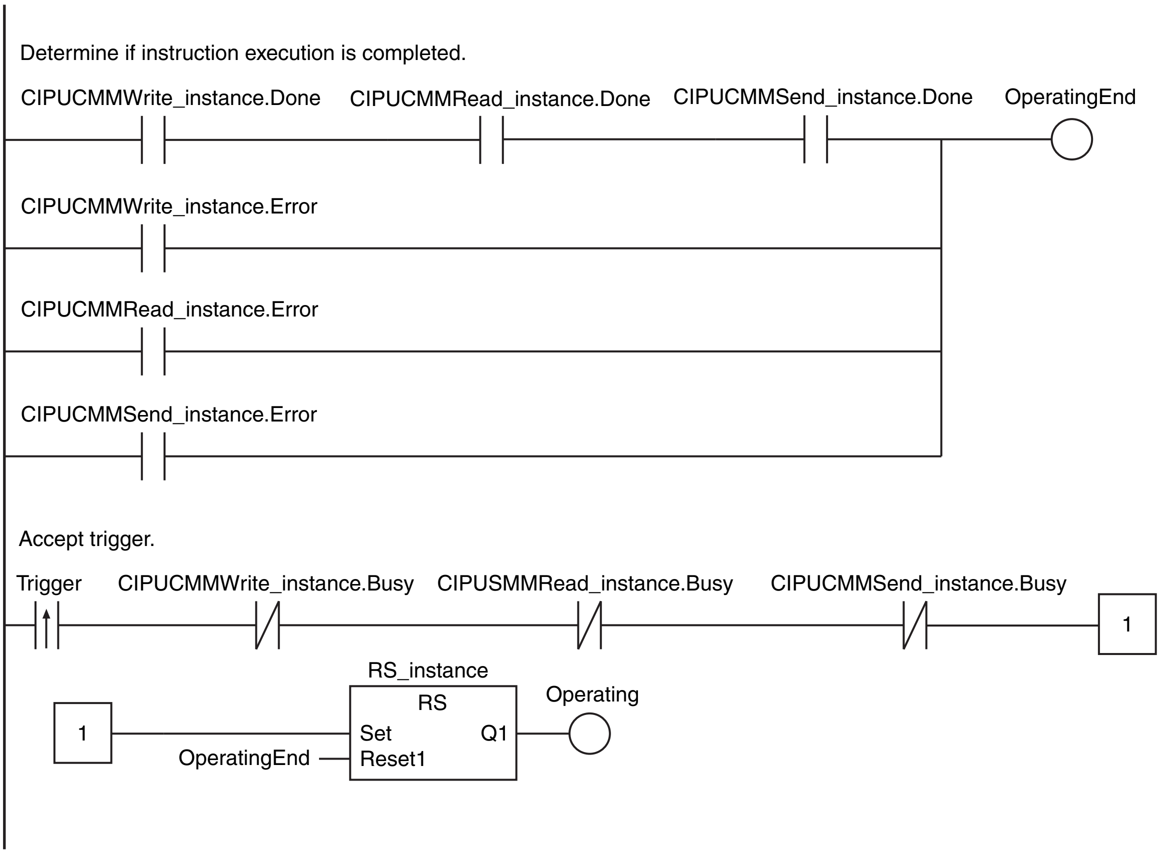
1. The CIPUCMMWrite instruction is used to write the value of a variable at a remote node. The variable name at the remote node is *WritingDat* and the contents of the *WriteDat* is written to it. *WritingDat* must be defined as a global variable at the remote node and the Network Publish attribute must be set.
2. The CIPUCMMRead instruction is used to read the value of a variable at a remote node. The value of the variable *OriginalDat* at the other node is read and the read value is stored in the *ReadDat* variable. *OriginalDat* must be defined as a global variable at the remote node and the Network Publish attribute must be set.
3. The CIPUCMMSend instruction is used to send an explicit message to a remote node. The contents of the message is to read identity information (product name). The class ID, instance ID, attribute ID, and service code are as follows: The response data is stored in the *ResDat* variable.

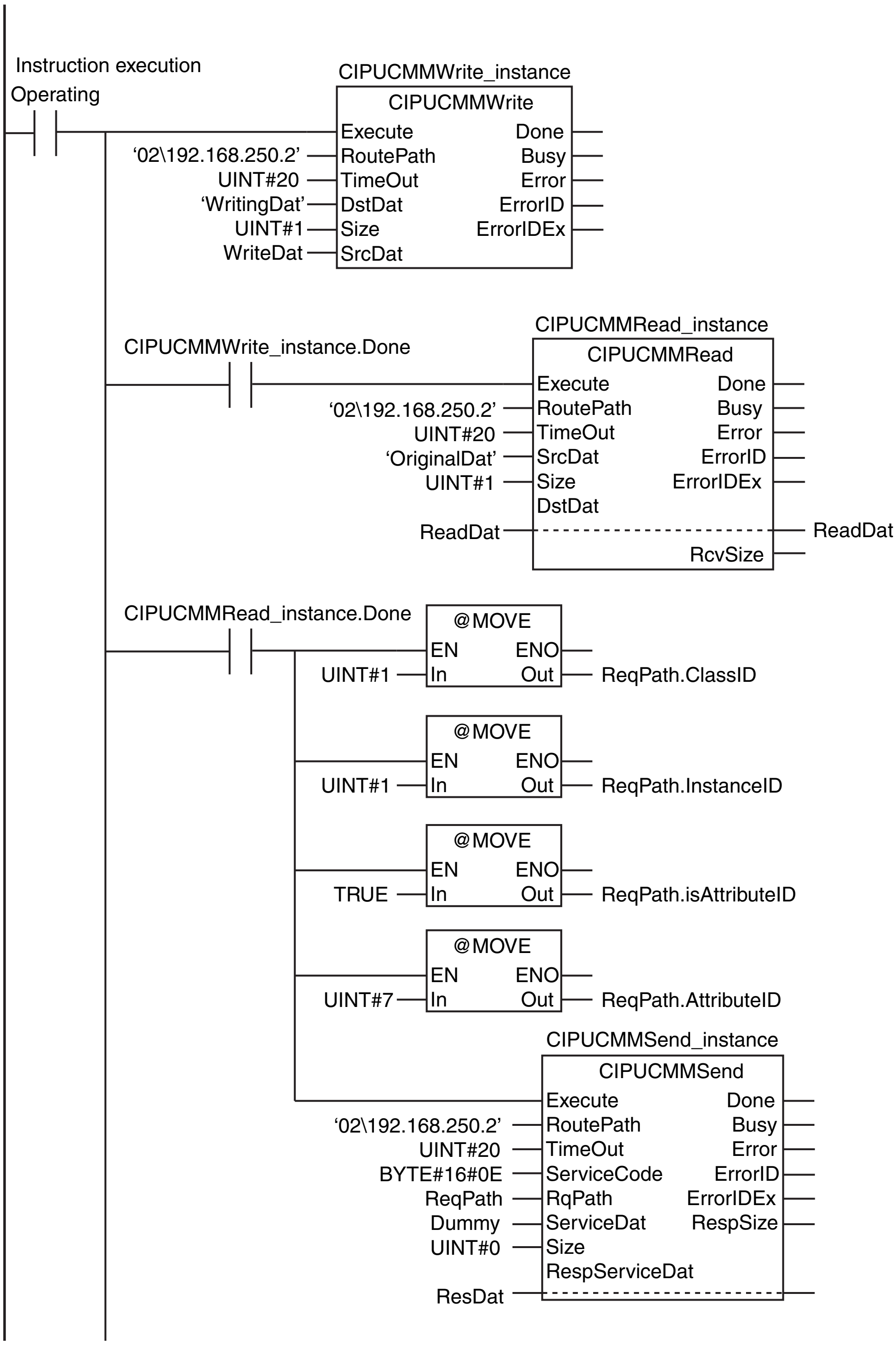
Item	Value
Class ID	1
Instance ID	1
Attribute ID	7
Service code	16#0E

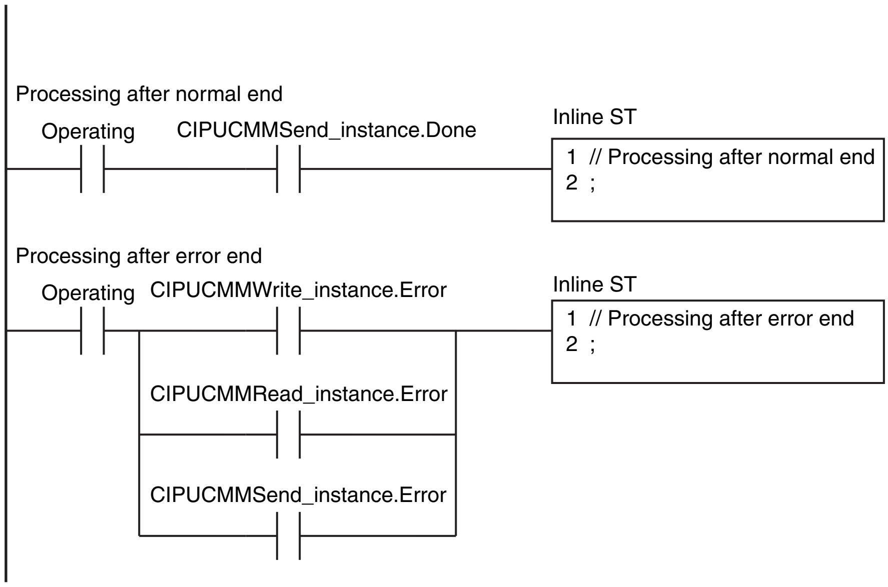


LD

Variable	Data type	Initial value	Comment
OperatingEnd	BOOL	FALSE	Processing completed
Trigger	BOOL	FALSE	Execution condition
Operating	BOOL	FALSE	Processing
WriteDat	INT	1234	Write data
ReadDat	INT	0	Read data
ReqPath	_sREQUEST_PATH	(ClassID:=0, InstanceID:=0, isAttributeID:=FALSE, AttributeID:=0)	Request path
ResDat	ARRAY[0..10] OF BYTE	[11(16#0)]	Response data
Dummy	BYTE	16#0	Dummy
RS_instance	RS		
CIPUCMMWrite_instance	CIPUCMMWrite		
CIPUCMMRead_instance	CIPUCMMRead		
CIPUCMMSend_instance	CIPUCMMSend		




Instruction execution
Operating
CIPUCMMWrite_instance
CIPUCMMWrite
Execute
RoutePath
TimeOut
DstDat
Size
SrcDat
Done
Busy
Error
ErrorID
ErrorIDEx
'02\192.168.250.2'
UINT#20
'WritingDat'
UINT#1
WriteDat
CIPUCMMWrite_instance.Done
CIPUCMMRead_instance
CIPUCMMRead
Execute
RoutePath
TimeOut
SrcDat
Size
DstDat
Done
Busy
Error
ErrorID
ErrorIDEx
RcvSize
'02\192.168.250.2'
UINT#20
'OriginalDat'
UINT#1
ReadDat
ReadDat
CIPUCMMRead_instance.Done
@MOVE
EN
ENO
In
Out
UINT#1
ReqPath.ClassID
@MOVE
EN
ENO
In
Out
UINT#1
ReqPath.InstanceID
@MOVE
EN
ENO
In
Out
TRUE
ReqPath.isAttributeID
@MOVE
EN
ENO
In
Out
UINT#7
ReqPath.AttributeID
CIPUCMMSend_instance
CIPUCMMSend
Execute
RoutePath
TimeOut
ServiceCode
RqPath
ServiceDat
Size
RespServiceDat
Done
Busy
Error
ErrorID
ErrorIDEx
RespSize
'02\192.168.250.2'
UINT#20
BYTE#16#0E
ReqPath
Dummy
UINT#0
ResDat



ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	FALSE	Execution condition
	DoUCMMTrigger	BOOL	FALSE	Processing
	Stage	INT	0	Stage change
	WriteDat	INT	0	Write data
	ReadDat	INT	0	Read data
	ReqPath	_sREQUEST_PATH	(ClassID:=0, InstanceID:=0, isAttributeID:=FALSE, AttributeID:=0)	Request path
	ResDat	ARRAY[0..10] OF BYTE	[11(16#0)]	Response data
	Dummy	BYTE	16#0	Dummy
	CIPUCMMWrite_instance	CIPUCMMWrite		
	CIPUCMMRead_instance	CIPUCMMRead		
	CIPUCMMSend_instance	CIPUCMMSend		

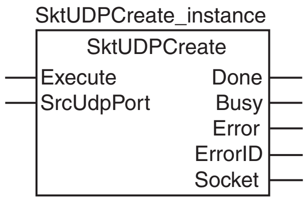
External Variables	Variable	Constant	Data type	Comment
	_EIP_EtnOnlineSta	✔	BOOL	Online

```
// Start sequence when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (DoUCMMTrigger=FALSE) AND (_Eip_EtnOnlineSta=TRUE) ) THEN
    DoUCMMTrigger:=TRUE;
    Stage        :=INT#1;
    CIPUCMMWrite_instance(
        Execute      :=FALSE,       // Initialize instance.
        SrcDat       :=WriteDat);   // Dummy
    CIPUCMMRead_instance(           // Initialize instance.
        Execute      :=FALSE,       // Dummy
        DstDat       :=ReadDat);    // Dummy
    CIPUCMMSend_instance(
        Execute      :=FALSE,       // Initialize instance.
        ServiceDat   := Dummy,      // Dummy
        RespServiceDat:=ResDat);    // Dummy
```

```
END_IF;

IF (DoUCMMTrigger=TRUE) THEN
    CASE Stage OF
    1 :     // Request writing value of variable.
        CIPUCMMWrite_instance(
            Execute  :=TRUE,
            RoutePath:='02\192.168.250.2',   // Route path
            TimeOut  :=UINT#20,              // Timeout time
            DstDat   :='WritingDat',         // Destination variable name
            Size     :=UINT#1,               // Number of elements to write
            SrcDat   :=WriteDat);            // Write data

    IF (CIPUCMMWrite_instance.Done=TRUE) THEN
        Stage:=INT#2;    // Normal end
    ELSIF (CIPUCMMWrite_instance.Error=TRUE) THEN
        Stage:=INT#10;   // Error end
            END_IF;

    2 :     // Request reading value of variable.
        CIPUCMMRead_instance(
            Execute  :=TRUE,
            RoutePath:='02\192.168.250.2',  // Route path
            TimeOut  :=UINT#20,             // Timeout time
            SrcDat   :='OriginalDat',       // Destination variable name
            Size     :=UINT#1,              // Number of elements to read
            DstDat   :=ReadDat);            // Read data

        IF (CIPUCMMRead_instance.Done=TRUE) THEN
            Stage:=INT#3;    // Normal end
        ELSIF (CIPUCMMRead_instance.Error=TRUE) THEN
            Stage:=INT#40;   // Error end
        END_IF;

    3 :     // Send message
        ReqPath.ClassID      :=UINT#01;
        ReqPath.InstanceID   :=UINT#01;
        ReqPath.isAttributeID:=TRUE;
        ReqPath.AttributeID  :=UINT#07;
        CIPUCMMSend_instance(
            Execute       :=TRUE,
            RoutePath     :='02\192.168.250.2',   // Route path
            TimeOut       :=UINT#20,              // Timeout time
            ServiceCode   :=BYTE#16#0E,           // Service code
            RqPath        :=ReqPath,              // Request path
            ServiceDat    :=Dummy,                // Service data
            Size          :=UINT#0,               // Number of elements
            RespServiceDat:=ResDat);              // Response data

        IF (CIPUCMMSend_instance.Done=TRUE) THEN
            Stage:=INT#0;     // Normal end
        ELSIF (CIPUCMMSend_instance.Error=TRUE) THEN
            Stage:=INT#30;    // Error end
        END_IF;

    0:     // Processing after normal end
        DoUCMMTrigger:=FALSE;
        Trigger      :=FALSE;

    ELSE   // Processing after error end
        DoUCMMTrigger:=FALSE;
        Trigger      :=FALSE;
    END_CASE;
END_IF;
```

SktUDPCreate

The SktUDPCreate instruction creates a UDP socket request to open a servo port for the built-in Ether-Net/IP.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
SktUDP Create	Create UDP Socket	FB		SktUDPCreate_instance(Execute, SrcUdpPort, Done, Busy, Error, ErrorID, Socket);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
SrcUdpPort	Local UDP port number	Input	Local UDP port number	1 to 65535	---	1
Socket	Socket	Output	Socket	---	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
SrcUdpPort							OK													
Socket	Refer to *Function* for details on the structure _sSOCKET.																			

Function

The SktUDPCreate instruction opens the port specified with the local UDP port number *ScrUdpPort*. To do this, it executes the Socket() and Bind() socket functions. Information on the socket that is opened is stored in *Socket*. The UDP port is open when the instruction is completed normally (i.e., when the value of *Done* changes to TRUE).

The data type of *Socket* is structure _sSOCKET. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Socket	Socket	Socket	_sSOCKET	---	---	---
Handle	Handle	Handle for data communications	UDINT	Depends on data type.	---	0
SrcAdr*1	Local address	Local IP address and port number	_sSOCKET_ADDRESS	---	---	---
PortNo	Port number	Port number	UINT	1 to 65535	---	0
IpAdr*1	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		''
DstAdr*1	Destination address	Destination IP address and port number	_sSOCKET_ADDRESS	---	---	---
PortNo*1	Port number	Port number	UINT	1 to 65535	---	0
IpAdr*1	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		''

*1 A value of 0 or NULL is output for these members.

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta*1 _EIP1_EtnOnlineSta*2 _EIP2_EtnOnlineSta*3 _EIPIn1_EtnOnlineSta*4	Online	BOOL	This variable indicates when built-in EtherNet/IP port communications can be used. TRUE: Communications are possible. FALSE: Communications are not possible.

*1 Use this variable name for an NJ-series CPU Unit.

*2 Use this variable name for port 1 on an NX-series CPU Unit, or for an NY-series Controller.

*3 Use this variable name for port 2 on an NX-series CPU Unit.

*4 Use this variable name for the internal communication port on an NY-series Controller.

Additional Information

Refer to the *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User's Manual* (Cat. No. W506) or *NY-series Industrial Panel PC / Industrial Box PC Built-in EtherNet/IP Port User's Manual* (Cat. No. W563) for details on socket services.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the built-in EtherNet/IP on NJ/NX-series CPU Units and NY-series Controllers.
- Use the SktClose instruction to close handles that are created with this instruction.
- Handles that are created with this instruction are disabled when you change to PROGRAM mode.

- You can execute a maximum of 32 of the following instructions at the same time: SktUDPCreate, SktUDPRcv, SktUDPSend, SktTCPAccept, SktTCPConnect, SktTCPRcv, SktTCPSend, SktGetTCP-Status, SktClose, SktClearBuf, and SktSetOption.
- An error occurs in the following cases. *Error* will change to TRUE.
  - There is a setting error for the local IP address.
  - The value of *SrcUdpPort* is outside of the valid range.
  - The port that is specified with *SrcUdpPort* is already open or close processing is in progress for it.
  - The port that is specified with *ScrUdpPort* is already in use.

Version Information

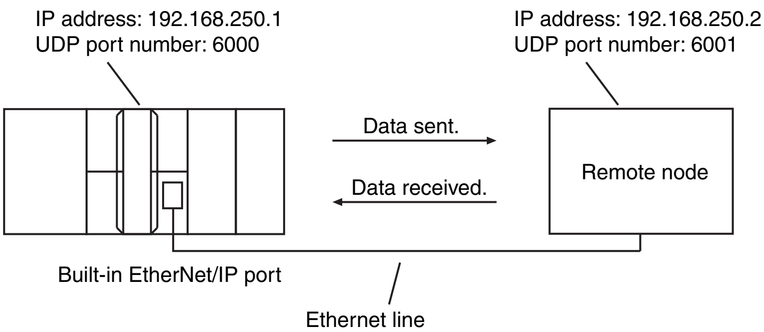
- The number of sockets that you can open at the same time depends on the unit version of the CPU Unit as shown in the following table. These limits are the totals for both UDP and TCP sockets.

Unit version of CPU Unit	Number of sockets
1.03 or higher	30 max.
1.02 or lower	16 max.

- For CPU Unit version 1.10 or later, the value of *Socket* does not change even if *Error* changes to TRUE. For version 1.09 or earlier, the value of *Socket* changes to 0.

Sample Programming

In this sample, the UDP socket service is used for data communications between the NJ/NX-series Controller and a remote node.



The processing procedure is as follows:

1. The SktUDPCreate instruction is used to request creating a UDP socket.
2. The SktUDPSend instruction is used to request sending data. The data in *SendSocketDat[]* is sent.
3. The SktUDPRcv instruction is used to request receiving data. The received data is stored in *RcvSocketDat[]*.
4. The SktClose instruction is used to close the socket.

ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	FALSE	Execution condition
	DoSendAndRcv	BOOL	FALSE	Processing
	Stage	INT	0	Stage change
	RcvSocketDat	ARRAY[0..1999] OF BYTE	[2000(16#0)]	Receive data
	WkSocket	_sSOCKET	(Handle:=0, SrcAdr:=(PortNo:=0, IpAdr:=''), DstAdr:=(PortNo:=0, IpAdr:=''))	Socket
	SendSocketDat	ARRAY[0..1999] OF BYTE	[2000(16#0)]	Send data
	SktUDPCreate_instance	SktUDPCreate		
	SktUDPSend_instance	SktUDPSend		
	SktUDPRcv_instance	SktUDPRcv		
	SktClose_instance	SktClose		

External Variables	Variable	Data type	Constant	Comment
	_EIP_EtnOnlineSta	BOOL	✔	Online

```
// Start sequence when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (DoSendAndRcv=FALSE) AND (_Eip_EtnOnlineSta=TRUE) ) THEN
    DoSendAndRcv:=TRUE;
    Stage       :=INT#1;
    SktUDPCreate_instance(Execute:=FALSE);          // Initialize instance.
    SktUDPSend_instance(                            // Initialize instance.
            Execute        :=FALSE,
            SendDat        :=SendSocketDat[0]);   // Dummy
    SktUDPRcv_instance(                             // Initialize instance.
            Execute        :=FALSE,
            RcvDat         :=RcvSocketDat[0]);    // Dummy
    SktClose_instance(Execute:=FALSE);              // Initialize instance.
END_IF;


IF (DoSendAndRcv=TRUE) THEN
    CASE Stage OF
    1 :     // Request creating socket.
        SktUDPCreate_instance(
            Execute    :=TRUE,
            SrcUdpPort:=UINT#6000,    // Local UDP port number
            Socket     =>WkSocket);   // Socket

        IF (SktUDPCreate_instance.Done=TRUE) THEN
            Stage:=INT#2;    // Normal end
        ELSIF (SktUDPCreate_instance.Error=TRUE) THEN
            Stage:=INT#10;   // Error end
        END_IF;

    2 :     // Request sending data
        WkSocket.DstAdr.PortNo:=UINT#6001;
        WkSocket.DstAdr.IpAdr :='192.168.250.2';
        SktUDPSend_instance(
            Execute:=TRUE,
            Socket :=WkSocket,          // Socket
            SendDat:=SendSocketDat[0],  // Send data
            Size   :=UINT#2000);        // Send data size
```

```
        IF (SktUDPSend_instance.Done=TRUE) THEN
            Stage:=INT#3;    // Normal end
        ELSIF (SktUDPSend_instance.Error=TRUE) THEN
            Stage:=INT#20;   // Error end
        END_IF;

    3 :     // Request receiving data.
        SktUDPRcv_instance(
            Execute:=TRUE,
            Socket :=WkSocket,          // Socket
            TimeOut:=UINT#0,            // Timeout time
            Size   :=UINT#2000,         // Receive data size
            RcvDat :=RcvSocketDat[0]);  // Receive data

        IF (SktUDPRcv_instance.Done=TRUE) THEN
            Stage:=INT#4;  // Normal end
        ELSIF (SktUDPRcv_instance.Error=TRUE) THEN
            Stage:=INT#30; // Error end
        END_IF;

    4 :     // Request closing.
        SktClose_instance(
            Execute:=TRUE,
            Socket :=WkSocket);  // Socket

        IF (SktClose_instance.Done=TRUE) THEN
            Stage:=INT#0;        // Normal end
        ELSIF (SktClose_instance.Error=TRUE) THEN
            Stage:=INT#40;       // Error end
        END_IF;

    0 :     // Normal end
        DoSendAndRcv:=FALSE;
        Trigger     :=FALSE;

    ELSE    // Interrupted by error.
        DoSendAndRcv:=FALSE;
        Trigger     :=FALSE;
    END_CASE;

END_IF;
```

● Programming in the Remote Node

In this example, programming is also required in the remote node. The order of sending and receiving is reversed in comparison with the above procedure.

1. The SktUDPCreate instruction is used to request creating a UDP socket.
2. The SktUDPRcv instruction is used to request receiving data. The received data is stored in *RcvSocketDat[]*.
3. The SktUDPSend instruction is used to request sending data. The data in *SendSocketDat[]* is sent.
4. The SktClose instruction is used to close the socket.

ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	FALSE	Execution condition
	DoSendAndRcv	BOOL	FALSE	Processing
	Stage	INT	0	Stage change
	RcvSocketDat	ARRAY[0..1999] OF BYTE	[2000(16#0)]	Receive data
	WkSocket	_sSOCKET	(Handle:=0, SrcAdr:=(PortNo:=0, IpAdr:=''), DstAdr:=(PortNo:=0, IpAdr:=''))	Socket
	SendSocketDat	ARRAY[0..1999] OF BYTE	[2000(16#0)]	Send data
	SktUDPCreate_instance	SktUDPCreate		
	SktUDPSend_instance	SktUDPSend		
	SktUDPRcv_instance	SktUDPRcv		
	SktClose_instance	SktClose		

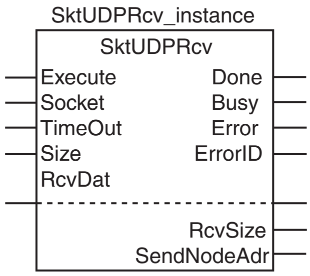
External Variables	Variable	Data type	Constant	Comment
	_EIP_EtnOnlineSta	BOOL	✔	Online

```
// Start sequence when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (DoSendAndRcv=FALSE) AND (_Eip_EtnOnlineSta=TRUE) ) THEN
    DoSendAndRcv:=TRUE;
    Stage       :=INT#1;
    SktUDPCreate_instance(Execute:=FALSE);    // Initialize instance.
    SktUDPSend_instance(                      // Initialize instance.
        Execute :=FALSE,
        SendDat :=SendSocketDat[0]);   // Dummy
    SktUDPRcv_instance(                       // Initialize instance.
        Execute :=FALSE,
        RcvDat  :=RcvSocketDat[0]);    // Dummy
    SktClose_instance(Execute:=FALSE);        // Initialize instance.
END_IF;


IF (DoSendAndRcv=TRUE) THEN
    CASE Stage OF
    1 :     // Request creating socket.
        SktUDPCreate_instance(
            Execute    :=TRUE,
            SrcUdpPort:=UINT#6001,    // Local UDP port number
            Socket     =>WkSocket);   // Socket

        IF (SktUDPCreate_instance.Done=TRUE) THEN
            Stage:=INT#2;  // Normal end
        ELSIF (SktUDPCreate_instance.Error=TRUE) THEN
            Stage:=INT#10; // Error end
        END_IF;

    2 :     // Request receiving data
        SktUDPRcv_instance(
            Execute:=TRUE,
            Socket :=WkSocket,           // Socket
            TimeOut:=UINT#0,             // Timeout time
            Size   :=UINT#2000,          // Receive data size
            RcvDat :=RcvSocketDat[0]);   // Receive data
```

```
        IF (SktUDPRcv_instance.Done=TRUE) THEN
            Stage:=INT#3;   // Normal end
        ELSIF (SktUDPRcv_instance.Error=TRUE) THEN
            Stage:=INT#20;  // Error end
        END_IF;

    3 :     // Request sending data.
        WkSocket.DstAdr.PortNo:=UINT#6000;
        WkSocket.DstAdr.IpAdr :='192.168.250.1';
        SktUDPSend_instance(
            Execute :=TRUE,
            Socket  :=WkSocket,         // Socket
            SendDat :=SendSocketDat[0], // Send data
            Size    :=UINT#2000);       // Send data size

        IF (SktUDPSend_instance.Done=TRUE) THEN
            Stage:=INT#4;   // Normal end
        ELSIF (SktUDPSend_instance.Error=TRUE) THEN
            Stage:=INT#30;  // Error end
        END_IF;

    4 :     // Request closing.
        SktClose_instance(
            Execute:=TRUE,
            Socket :=WkSocket);  // Socket

        IF (SktClose_instance.Done=TRUE) THEN
            Stage:=INT#0;        // Normal end
        ELSIF (SktClose_instance.Error=TRUE) THEN
            Stage:=INT#40;       // Error end
        END_IF;

    0 :     // Normal end
        DoSendAndRcv:=FALSE;
        Trigger     :=FALSE;

    ELSE    // Interrupted by error.
        DoSendAndRcv:=FALSE;
        Trigger     :=FALSE;
    END_CASE;

END_IF;
```

SktUDPRcv

The SktUDPRcv instruction reads the data from the receive buffer for a UDP socket for the built-in EtherNet/IP.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
SktUDPRcv	UDP Socket Receive	FB		SktUDPRcv_instance(Execute, Socket, TimeOut, Size, RcvDat, Done, Busy, Error, ErrorID, RcvSize, SendNodeAdr);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Socket	Socket	Input	Socket	---	---	---
TimeOut	Timeout time		0: No timeouts 1 to 65535: 0.1 to 6553.5s	Depends on data type.	0.1 s	0
Size	Stored size		The number of bytes to read from the receive buffer	0 to 2000	Bytes	1
RcvDat[] (array)	Receive data	In-out	Receive data	Depends on data type.	---	---
RcvSize	Receive data size	Output	The number of bytes actually stored in *RcvDat[]*	0 to 2000	Bytes	---
SendNodeAdr	Source node address		Source node address	---	---	

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Socket	Refer to *Function* for details on the structure _sSOCKET.																			
TimeOut							OK													
Size							OK													
RcvDat[] (array)		OK																		
RcvSize							OK													
SendNodeAdr	Refer to *Function* for details on the structure _sSOCKET_ADDRESS.																			

Function

The SktUDPRcv instruction stores the data in the receive buffer for the socket that is specified with *Socket* in receive data *RcvDat[]*. The number of bytes to store is specified with *Size*. The number of bytes that is actually stored is assigned to *RcvSize*. The node address of the node that sent the data is stored in *SendNodeAdr*.

If there is no data in the receive buffer, the instruction waits for data for the time that is set with timeout time *TimeOut*. Storage of the data to *RcvDat[]* is completed when the instruction is completed normally (i.e., when the value of *Done* changes to TRUE).

The data type of *Socket* is structure _sSOCKET. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Socket	Socket	Socket	_sSOCKET	---	---	---
Handle	Handle	Handle for data communications	UDINT	Depends on data type.	---	0
SrcAdr*	Local address	Local IP address and port number	_sSOCKET_ADDRESS	---	---	---
PortNo*	Port number	Port number	UINT	1 to 65535	---	0
IpAdr*	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		''
DstAdr*	Destination address	Destination IP address and port number	_sSOCKET_ADDRESS	---	---	---
PortNo*	Port number	Port number	UINT	1 to 65535	---	0
IpAdr*	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		''

* These members are not used for this instruction.

The data type of *SendNodeAdr* is structure _sSOCKET_ADDRESS. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
SendNodeAdr	Source node address	Source node address	_sSOCKET_ADDRESS	---	---	---
PortNo	Port number	UPD port number of the source node	UINT	1 to 65535	---	---
IpAdr	IP address	IP address of the source node	STRING	Depends on data type.		

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta*1	Online	BOOL	This variable indicates when built-in EtherNet/IP port communications can be used. TRUE: Communications are possible. FALSE: Communications are not possible.
_EIP1_EtnOnlineSta*2 _EIP2_EtnOnlineSta*3			
_EIPIn1_EtnOnlineSta*4			

*1 Use this variable name for an NJ-series CPU Unit.

*2 Use this variable name for port 1 on an NX-series CPU Unit, or for an NY-series Controller.

*3 Use this variable name for port 2 on an NX-series CPU Unit.

*4 Use this variable name for the internal communication port on an NY-series Controller.

Additional Information

Refer to the *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User's Manual* (Cat. No. W506) or *NY-series Industrial Panel PC / Industrial Box PC Built-in EtherNet/IP Port User's Manual* (Cat. No. W563) for details on socket services.

Precautions for Correct Use

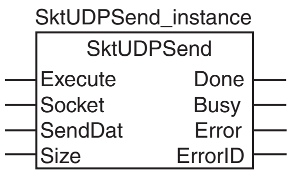
- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the built-in EtherNet/IP on NJ/NX-series CPU Units and NY-series Controllers.
- Up to 2,000 bytes of data can be read from the receive buffer with one instruction.
- If the size of data that was received by the specified socket is smaller than the value of *Size*, then all of the received data is stored in *RecDat[].* Then size of data that was stored is stored in *RcvSize*.
- If the size of data that was received by the specified socket is larger than the value of *Size*, then the size of received data specified by *Size* is stored in *RecDat[].*
- The receive data is not read if the value of *Size* is 0.
- If the SktClose instruction closes the connection when there is no data in the receive buffer, a normal end occurs without waiting to receive data even if a timeout has not occurred. The value of *RcvSize* is 0 in that case.
- You can execute a maximum of 32 of the following instructions at the same time: SktUDPCreate, SktUDPRcv, SktUDPSend, SktTCPAccept, SktTCPConnect, SktTCPRcv, SktTCPSend, SktGetTCP-Status, SktClose, SktClearBuf, and SktSetOption.
- An error occurs in the following cases. *Error* will change to TRUE.
  - There is a setting error for the local IP address.
  - Data reception is in progress for the socket specified with *Socket.*
  - The socket specified with *Socket* is not open.
  - The handle specified by *Socket.Handle* does not exist.

Sample Programming

Refer to the sample programming that is provided for the SktUDPCreate instruction (page 2-1053).

SktUDPSend

The SktUDPSend instruction sends data from a UDP port for the built-in EtherNet/IP.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
SktUDPSend	UDP Socket Send	FB		SktUDPSend_instance(Execute, Socket, SendDat, Size, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Socket	Socket	Input	Socket	---	---	---
SendDat[] (array)	Send data		Send data	Depends on data type.		
Size	Send data size		Send data size	0 to 2000	Bytes	1

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Socket	Refer to *Function* for details on the structure _sSOCKET.																			
SendDat[] (array)		OK																		
Size							OK													

Function

The SktUDPSend instruction sends send data *SendDat[]* from the socket that is specified with *Socket*. The number of bytes to send is specified with *Size*. The remote node is specified with *Socket.DstAdr*. Transmission of *SendDat[]* to the send buffer is completed when the instruction is completed normally (i.e., when the value of *Done* changes to TRUE).

The data type of *Socket* is structure _sSOCKET. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Socket	Socket	Socket	_sSOCKET	---	---	---
Handle	Handle	Handle for data communications	UDINT	Depends on data type.	---	0
SrcAdr*	Local address	Local IP address and port number	_sSOCKET_ ADDRESS	---	---	---
PortNo*	Port number	Port number	UINT	1 to 65535	---	0
IpAdr*	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		‘’
DstAdr	Destination address	Destination IP address and port number	_sSOCKET_ ADDRESS	---	---	---
PortNo	Port number	Port number	UINT	1 to 65535	---	0
IpAdr	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		‘’

* These members are not used for this instruction.

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta*1	Online	BOOL	This variable indicates when built-in EtherNet/IP port communications can be used. TRUE: Communications are possible. FALSE: Communications are not possible.
_EIP1_EtnOnlineSta*2 _EIP2_EtnOnlineSta*3			
_EIPIn1_EtnOnlineSta*4			

*1 Use this variable name for an NJ-series CPU Unit.

*2 Use this variable name for port 1 on an NX-series CPU Unit, or for an NY-series Controller.

*3 Use this variable name for port 2 on an NX-series CPU Unit.

*4 Use this variable name for the internal communication port on an NY-series Controller.

Additional Information

Refer to the *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User’s Manual* (Cat. No. W506) or *NY-series Industrial Panel PC / Industrial Box PC Built-in EtherNet/IP Port User’s Manual* (Cat. No. W563) for details on socket services.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the built-in EtherNet/IP on NJ/NX-series CPU Units and NY-series Controllers.
- Up to 2,000 bytes of data can be sent with one instruction. A maximum of 2,000 bytes is sent even if the *SendDat[]* array is larger than 2,000 bytes. Only 1,472 bytes can be sent if the broadcast address is specified.
- If the value of *Size* is 0, then 0 bytes of send data is transmitted on the line.

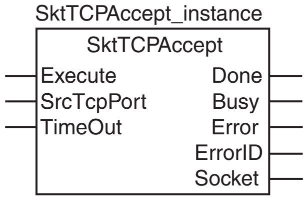
- You can execute a maximum of 32 of the following instructions at the same time: SktUDPCreate, SktUDPRcv, SktUDPSend, SktTCPAccept, SktTCPConnect, SktTCPRcv, SktTCPSend, SktGetTCP-Status, SktClose, SktClearBuf, and SktSetOption.
- An error occurs in the following cases. *Error* will change to TRUE.
  - There is a setting error for the local IP address.
  - The value of a member of *Socket* is outside of the valid range.
  - Data transmission is in progress for the socket specified with *Socket.*
  - The socket specified with *Socket* is not open.
  - The remote node for *Socket* was specified with a domain name and address resolution failed.
  - The handle specified by *Socket.Handle* does not exist.
  - The value of *Size* exceeds the number of elements in *SendDat[]*.

Sample Programming

Refer to the sample programming that is provided for the SktUDPCreate instruction (page 2-1053).

SktTCPAccept

The SktTCPAccept instruction requests accepting a TCP socket for the built-in EtherNet/IP.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
SktTCPAccept	Accept TCP Socket	FB		SktTCPAccept_instance(Execute, SrcTcpPort, TimeOut, Done, Busy, Error, ErrorID, Socket);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
SrcTcpPort	Local TCP port number	Input	Local TCP port number	1 to 65535	---	1
TimeOut	Timeout time	Input	0: No timeouts 1 to 65535: 0.1 to 6553.5s	Depends on data type.	0.1 s	0
Socket	Socket	Output	Socket	---	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
SrcTcpPort							OK													
TimeOut							OK													
Socket	Refer to *Function* for details on the structure _sSOCKET.																			

Function

The SktTCPAccept instruction requests accepting the port specified with the local TCP port number *ScrTcpPort*. To do this, it executes the Socket(), Bind(), Listen(), and Accept() socket functions. The instruction waits for the time set with timeout time *TimeOut* for a connection to be established with the remote node. The connection is established when the instruction is completed normally (i.e., when the value of *Done* changes to TRUE).

The data type of *Socket* is structure _sSOCKET. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Socket	Socket	Socket	_sSOCKET	---	---	---
Handle	Handle	Handle for data communications	UDINT	Depends on data type.	---	0
SrcAdr	Local address	Local IP address and port number	_sSOCKET_ADDRESS	---	---	---
PortNo	Port number	Port number	UINT	1 to 65535	---	0
IpAdr*1	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		''
DstAdr	Destination address	Destination IP address and port number	_sSOCKET_ADDRESS	---	---	---
PortNo	Port number	Port number	UINT	1 to 65535	---	0
IpAdr	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		''

*1 NULL is output for this member.

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta*1	Online	BOOL	This variable indicates when built-in EtherNet/IP port communications can be used. TRUE: Communications are possible. FALSE: Communications are not possible.
_EIP1_EtnOnlineSta*2 _EIP2_EtnOnlineSta*3			
_EIPIn1_EtnOnlineSta*4			

*1 Use this variable name for an NJ-series CPU Unit.

*2 Use this variable name for port 1 on an NX-series CPU Unit, or for an NY-series Controller.

*3 Use this variable name for port 2 on an NX-series CPU Unit.

*4 Use this variable name for the internal communication port on an NY-series Controller.

Additional Information

- Refer to the *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User's Manual* (Cat. No. W506) or *NY-series Industrial Panel PC / Industrial Box PC Built-in EtherNet/IP Port User's Manual* (Cat. No. W563) for details on socket services.
- You can execute this instruction more than once to open connections to more than one client with one local port number. A different socket is returned for each connection.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the built-in EtherNet/IP on NJ/NX-series CPU Units and NY-series Controllers.
- Use the SktClose instruction to close handles that are created with this instruction.
- Handles that are created with this instruction are disabled when you change to PROGRAM mode.
- You can execute a maximum of 32 of the following instructions at the same time: SktUDPCreate, SktUDPRcv, SktUDPSend, SktTCPAccept, SktTCPConnect, SktTCPRcv, SktTCPSend, SktGetTCP-Status, SktClose, SktClearBuf, and SktSetOption.
- An error occurs in the following cases. *Error* will change to TRUE.
  - There is a setting error for the local IP address.
  - The value of *SrcTcpPort* is outside of the valid range.
  - Open processing is in progress for the socket specified with *SrcTcpPort.*
  - Close processing is in progress for the socket specified with *SrcTcpPort.*
  - A connection is not opened within the time that is specified with *TimeOut*.

Version Information

- The number of sockets that you can open at the same time depends on the unit version of the CPU Unit as shown in the following table. These limits are the totals for both UDP and TCP sockets.

Unit version of CPU Unit	Number of sockets
1.03 or higher	30 max.
1.02 or lower	16 max.

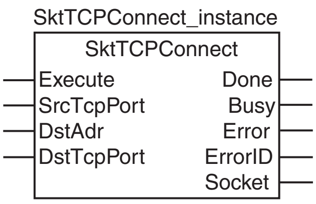
- For CPU Unit version 1.10 or later, the value of *Socket* does not change even if *Error* changes to TRUE. For version 1.09 or earlier, the value of *Socket* changes to 0.

Sample Programming

Refer to the sample programming that is provided for the SktTCPConnect instruction (page 2-1070).

SktTCPConnect

The SktTCPConnect instruction connects to a remote TCP port from the built-in EtherNet/IP.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
SktTCP Connect	Connect TCP Socket	FB		SktTCPConnect_instance(Execute, SrcTcpPort, DstAdr, DstTcpPort, Done, Busy, Error, ErrorID, Socket);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
SrcTcpPort	Local TCP port number	Input	Local TCP port number. If 0 is specified, an available TCP port that is 1024 or higher is automatically assigned. Well-known port numbers are not assigned.	Depends on data type.	---	0
DstAdr	Destination address		Destination IP address or host name	200 bytes max.		---
DstTcpPort	Destination TCP port number		Destination TCP port number	1 to 65,535		1
Socket	Socket	Output	Socket	---	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
SrcTcpPort							OK													
DstAdr																				OK
DstTcpPort							OK													
Socket	Refer to *Function* for details on the structure _sSOCKET.																			

Function

The SktTCPConnect instruction requests a connection between local TCP port number *SrcTcpPort* and destination TCP port number *DstTcpPort* at destination address *DstAdr*. To do this, it executes the Connect() socket function. The connection is established when the instruction is completed normally (i.e., when the value of *Done* changes to TRUE).

The data type of *Socket* is structure _sSOCKET. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Socket	Socket	Socket	_sSOCKET	---	---	---
Handle	Handle	Handle for data communications	UDINT	Depends on data type.	---	0
SrcAdr	Local address	Local IP address and port number	_sSOCKET_ ADDRESS	---	---	---
PortNo	Port number	Port number	UINT	1 to 65535	---	0
IpAdr*1	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		‘’
DstAdr	Destination address	Destination IP address and port number	_sSOCKET_ ADDRESS	---	---	---
PortNo	Port number	Port number	UINT	1 to 65535	---	0
IpAdr	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		‘’

*1 NULL is output for this member.

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta*1 _EIP1_EtnOnlineSta*2 _EIP2_EtnOnlineSta*3 _EIPIn1_EtnOnlineSta*4	Online	BOOL	This variable indicates when built-in EtherNet/IP port communications can be used. TRUE: Communications are possible. FALSE: Communications are not possible.

*1 Use this variable name for an NJ-series CPU Unit.

*2 Use this variable name for port 1 on an NX-series CPU Unit, or for an NY-series Controller.

*3 Use this variable name for port 2 on an NX-series CPU Unit.

*4 Use this variable name for the internal communication port on an NY-series Controller.

Additional Information

Refer to the *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User’s Manual* (Cat. No. W506) or *NY-series Industrial Panel PC / Industrial Box PC Built-in EtherNet/IP Port User’s Manual* (Cat. No. W563) for details on socket services.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the built-in EtherNet/IP on NJ/NX-series CPU Units and NY-series Controllers.
- Use the SktClose instruction to close handles that are created with this instruction.
- Handles that are created with this instruction are disabled when you change to PROGRAM mode.
- You can execute a maximum of 32 of the following instructions at the same time: SktUDPCreate, SktUDPRcv, SktUDPSend, SktTCPAccept, SktTCPConnect, SktTCPRcv, SktTCPSend, SktGetTCP-Status, SktClose, SktClearBuf, and SktSetOption.
- An error occurs in the following cases. *Error* will change to TRUE.
  - There is a setting error for the local IP address.
  - The value of *DstAdr* is outside of the valid range.
  - The value of *DstTcpPort* is outside of the valid range.
  - The TCP port that is specified with *SrcTcpPort* is already open.
  - The remote node that is specified with *DstAdr* does not exist.
  - The remote node that is specified with *DstAdr* and *DstTcpPort* is not waiting for a connection.
  - Address resolution failed for the host name that is specified with *DstAdr*.
  - A connection is already open for the same client (IP address and TCP port).

Version Information

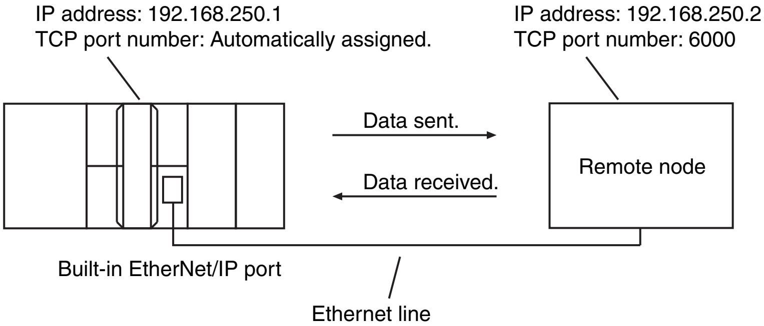
- The number of sockets that you can open at the same time depends on the unit version of the CPU Unit as shown in the following table. These limits are the totals for both UDP and TCP sockets.

Unit version of CPU Unit	Number of sockets
1.03 or higher	30 max.
1.02 or lower	16 max.

- For CPU Unit version 1.10 or later, the value of *Socket* does not change even if *Error* changes to TRUE. For version 1.09 or earlier, the value of *Socket* changes to 0.

Sample Programming

In this sample, the TCP socket service is used for data communications between the NJ/NX-series Controller and a remote node.



The processing procedure is as follows:

1. The SktTCPConnect instruction is used to request connecting to the TCP port on the remote node.
2. The SktClearBuf instruction is used to clear the receive buffer for a TCP socket.
3. The SktGetTCPStatus instruction is used to read the status of a TCP socket.
4. The SktTCPSend instruction is used to request sending data. The data in *SendSocketDat[]* is sent.
5. The SktTCPRcv instruction is used to request receiving data. The received data is stored in *RcvSocketDat[]*.
6. The SktClose instruction is used to close the socket.

ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	FALSE	Execution condition
	DoTCP	BOOL	FALSE	Processing
	Stage	INT	0	Stage change
	RcvSocketDat	ARRAY[0..1999] OF BYTE	[2000(16#0)]	Receive data
	WkSocket	_sSOCKET	(Handle:=0, SrcAdr:=(PortNo:=0, IpAdr:=''), DstAdr:=(PortNo:=0, IpAdr:=''))	Socket
	SendSocketDat	ARRAY[0..1999] OF BYTE	[2000(16#0)]	Send data
	SktTCPConnect_instance	SktTCPConnect		
	SktClearBuf_instance	SktClearBuf		
	SktGetTCPStatus_instance	SktGetTCPStatus		
	SktTCPSend_instance	SktTCPSend		
	SktTCPRcv_instance	SktTCPRcv		
	SktClose_instance	SktClose		

External Variables	Variable	Data type	Constant	Comment
	_EIP_EtnOnlineSta	BOOL	✔	Online

```
// Start sequence when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (DoTCP=FALSE) AND (_Eip_EtnOnlineSta=TRUE) ) THEN
    DoTCP :=TRUE;
    Stage :=INT#1;
    SktTCPConnect_instance(Execute:=FALSE);    // Initialize instance.
     SktClearBuf_instance(Execute:=FALSE);     // Initialize instance.
     SktGetTCPStatus_instance(Execute:=FALSE); // Initialize instance.
     SktTCPSend_instance(                      // Initialize instance.
        Execute:=FALSE,
          SendDat:=SendSocketDat[0]);          // Dummy
     SktTCPRcv_instance(                       // Initialize instance.
          Execute:=FALSE,
          RcvDat :=RcvSocketDat[0]);           // Dummy
     SktClose_instance(Execute:=FALSE);        // Initialize instance.
END_IF;


IF (DoTCP=TRUE) THEN
     CASE Stage OF
    1 :    // Request a connection.
        SktTCPConnect_instance(
            Execute   :=TRUE,
               SrcTcpPort:=UINT#0,          // Local TCP port number: Automatically assigned.
               DstAdr    :='192.168.250.2', // Remote IP address
               DstTcpPort:=UINT#6000,       // Destination TCP port number
               Socket     =>WkSocket);      // Socket

          IF (SktTCPConnect_instance.Done=TRUE) THEN
               Stage:=INT#2;     // Normal end
          ELSIF (SktTCPConnect_instance.Error=TRUE) THEN
               Stage:=INT#10;    // Error end
          END_IF;
```

```
2 :    // Clear receive buffer.
    SktClearBuf_instance(
        Execute:=TRUE,
        Socket :=WkSocket);   // Socket

    IF (SktClearBuf_instance.Done=TRUE) THEN
        Stage:=INT#3;         // Normal end
    ELSIF (SktClearBuf_instance.Error=TRUE) THEN
        Stage:=INT#20;        // Error end
    END_IF;

3 :    // Request reading status.
    SktGetTCPStatus_instance(
        Execute:=TRUE,
        Socket :=WkSocket);   // Socket

    IF (SktGetTCPStatus_instance.Done=TRUE) THEN
        Stage:=INT#4;         // Normal end
    ELSIF (SktGetTCPStatus_instance.Error=TRUE) THEN
        Stage:=INT#30;        // Error end
    END_IF;

4 :    // Request sending data
    SktTCPSend_instance(
        Execute:=TRUE,
        Socket :=WkSocket,         // Socket
        SendDat:=SendSocketDat[0], // Send data
        Size   :=UINT#2000);       // Send data size

    IF (SktTCPSend_instance.Done=TRUE) THEN
        Stage:=INT#5;              // Normal end
    ELSIF (SktTCPSend_instance.Error=TRUE) THEN
        Stage:=INT#40;             // Error end
    END_IF;

5 :    // Request receiving data
    SktTCPRcv_instance(
        Execute:=TRUE,
        Socket :=WkSocket,         // Socket
        TimeOut:=UINT#0,           // Timeout time
        Size   :=UINT#2000,        // Receive data size
        RcvDat :=RcvSocketDat[0]); // Receive data

    IF (SktTCPRcv_instance.Done=TRUE) THEN
        Stage:=INT#6;              // Normal end
    ELSIF (SktTCPRcv_instance.Error=TRUE) THEN
        Stage:=INT#50;             // Error end
    END_IF;

6 :    // Request closing.
    SktClose_instance(
        Execute:=TRUE,
        Socket :=WkSocket);        // Socket

    IF (SktClose_instance.Done=TRUE) THEN
        Stage:=INT#0;              // Normal end
    ELSIF (SktClose_instance.Error=TRUE) THEN
        Stage:=INT#40;             // Error end
    END_IF;

0 :    // Normal end
    DoTCP   :=FALSE;
    Trigger :=FALSE;
```

```
    ELSE   // Interrupted by error.
        DoTCP   :=FALSE;
        Trigger :=FALSE;
    END_CASE;

END_IF;
```

Programming in the Remote Node

In this example, programming is also required in the remote node. The order of sending and receiving is reversed in comparison with the above procedure.

1. The SktTCPAccept instruction is used to request accepting a TCP socket.
2. The SktTCPRcv instruction is used to request receiving data. The received data is stored in *RcvSocketDat[]*.
3. The SktTCPSend instruction is used to request sending data. The data in *SendSocketDat[]* is sent.
4. The SktClose instruction is used to close the socket.

ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	FALSE	Execution condition
	DoTCP	BOOL	FALSE	Processing
	Stage	INT	0	Stage change
	RcvSocketDat	ARRAY[0..1999] OF BYTE	[2000(16#0)]	Receive data
	WkSocket	_sSOCKET	(Handle:=0, SrcAdr:=(PortNo:=0, IpAdr:=''), DstAdr:=(PortNo:=0, IpAdr:=''))	Socket
	SendSocketDat	ARRAY[0..1999] OF BYTE	[2000(16#0)]	Send data
	SktTCPAccept_instance	SktTCPAccept		
	SktTCPSend_instance	SktTCPSend		
	SktTCPRcv_instance	SktTCPRcv		
	SktClose_instance	SktClose		

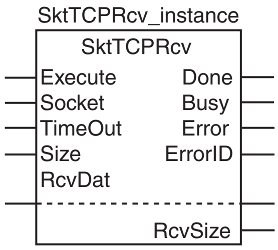
External Variables	Variable	Data type	Constant	Comment
	_EIP_EtnOnlineSta	BOOL	✔	Online

```
// Start sequence when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (DoTCP=FALSE) AND (_Eip_EtnOnlineSta=TRUE) ) THEN
    DoTCP:=TRUE;
    Stage:=INT#1;
    SktTCPAccept_instance(Execute:=FALSE);   // Initialize instance.
    SktTCPSend_instance(                     // Initialize instance.
        Execute :=FALSE,
        SendDat :=SendSocketDat[0]);         // Dummy
    SktTCPRcv_instance(                      // Initialize instance.
        Execute :=FALSE,
        RcvDat  :=RcvSocketDat[0]);          // Dummy
    SktClose_instance(Execute:=FALSE);       // Initialize instance.
END_IF;


IF (DoTCP=TRUE) THEN
    CASE Stage OF
    1 :     // Request accepting a socket connection.
        SktTCPAccept_instance(
            Execute    :=TRUE,
            SrcTcpPort:=UINT#6000,    // Local TCP port number
            TimeOut    :=UINT#0,      // Timeout time
            Socket     =>WkSocket);   // Socket

        IF (SktTCPAccept_instance.Done=TRUE) THEN
            Stage:=INT#2;             // Normal end
        ELSIF (SktTCPAccept_instance.Error=TRUE) THEN
            Stage:=INT#10;            // Error end
        END_IF;

    2 :     // Request receiving data
        SktTCPRcv_instance(
            Execute:=TRUE,
            Socket :=WkSocket,           // Socket
            TimeOut:=UINT#0,             // Timeout time
            Size   :=UINT#2000,          // Receive data size
            RcvDat :=RcvSocketDat[0]);   // Receive data
```

```
        IF (SktTCPRcv_instance.Done=TRUE) THEN
            Stage:=INT#3;     // Normal end
        ELSIF (SktTCPRcv_instance.Error=TRUE) THEN
            Stage:=INT#20;    // Error end
        END_IF;

    3 :     // Request sending data.
        SendSocketDat:=RcvSocketDat;
        SktTCPSend_instance(
            Execute:=TRUE,
            Socket :=WkSocket,         // Socket
            SendDat:=SendSocketDat[0], // Send data
            Size   :=UINT#2000);       // Send data size

        IF (SktTCPSend_instance.Done=TRUE) THEN
            Stage:=INT#4;   // Normal end
        ELSIF (SktTCPSend_instance.Error=TRUE) THEN
            Stage:=INT#30;  // Error end
        END_IF;

    4 :     // Request closing.
        SktClose_instance(
            Execute:=TRUE,
            Socket :=WkSocket);   // Socket

        IF (SktClose_instance.Done=TRUE) THEN
            Stage:=INT#0;         // Normal end
        ELSIF (SktClose_instance.Error=TRUE) THEN
            Stage:=INT#40;        // Error end
        END_IF;

    0 :     // Normal end
        DoTCP  :=FALSE;
        Trigger:=FALSE;

    ELSE    // Interrupted by error.
        DoTCP  :=FALSE;
        Trigger:=FALSE;
    END_CASE;

END_IF;
```

SktTCPRcv

The SktTCPRcv instruction reads the data from the receive buffer for a TCP socket for the built-in EtherNet/IP.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
SktTCPRcv	TCP Socket Receive	FB		SktTCPRcv_instance(Execute, Socket, TimeOut, Size, RcvDat, Done, Busy, Error, ErrorID, RcvSize);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Socket	Socket	Input	Socket	---	---	---
TimeOut	Timeout time		0: No timeouts 1 to 65535: 0.1 to 6553.5s	Depends on data type.	0.1 s	0
Size	Stored size		The number of bytes to read from the receive buffer	0 to 2000	Bytes	1
RcvDat[] (array)	Receive data	In-out	Receive data	Depends on data type.	---	---
RcvSize	Receive data size	Output	The number of bytes actually stored in *RcvDat[]*	1 to 2000	Bytes	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Socket	Refer to *Function* for details on the structure _sSOCKET.																			
TimeOut							OK													
Size							OK													
RcvDat[] (array)		OK																		
RcvSize							OK													

Function

The SktTCPRcv instruction stores the data in the receive buffer for the socket that is specified with *Socket* in receive data *RcvDat[].* The number of bytes to store is specified with *Size*. The number of bytes that is actually stored is assigned to *RcvSize*. If there is no data in the receive buffer, the instruction waits for data for the time that is set with timeout time *TimeOut*. Storage of the data to *RcvDat[]* is completed when the instruction is completed normally (i.e., when the value of *Done* changes to TRUE).

The data type of *Socket* is structure _sSOCKET. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Socket	Socket	Socket	_sSOCKET	---	---	---
Handle	Handle	Handle for data communications	UDINT	Depends on data type.	---	0
SrcAdr*	Local address	Local IP address and port number	_sSOCKET_ADDRESS	---	---	---
PortNo*	Port number	Port number	UINT	1 to 65535	---	0
IpAdr*	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		‘’
DstAdr*	Destination address	Destination IP address and port number	_sSOCKET_ADDRESS	---	---	---
PortNo*	Port number	Port number	UINT	1 to 65535	---	0
IpAdr*	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		‘’

* These members are not used for this instruction.

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta*1	Online	BOOL	This variable indicates when built-in EtherNet/IP port communications can be used. TRUE: Communications are possible. FALSE: Communications are not possible.
_EIP1_EtnOnlineSta*2			
_EIP2_EtnOnlineSta*3			
_EIPIn1_EtnOnlineSta*4			

*1 Use this variable name for an NJ-series CPU Unit.

*2 Use this variable name for port 1 on an NX-series CPU Unit, or for an NY-series Controller.

*3 Use this variable name for port 2 on an NX-series CPU Unit.

*4 Use this variable name for the internal communication port on an NY-series Controller.

Additional Information

Refer to the *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User’s Manual* (Cat. No. W506) or *NY-series Industrial Panel PC / Industrial Box PC Built-in EtherNet/IP Port User’s Manual* (Cat. No. W563) for details on socket services.

Precautions for Correct Use

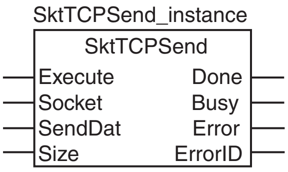
- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the built-in EtherNet/IP on NJ/NX-series CPU Units and NY-series Controllers.
- Up to 2,000 bytes of data can be read with one instruction. A maximum of 2,000 bytes is read even if the *RcvDat[]* array is larger than 2,000 bytes.
- If the size of data that was received by the specified socket is smaller than the value of *Size*, then all of the received data is stored in *RecDat[]*. Then size of data that was stored is stored in *RcvSize*.
- If the size of data that was received by the specified socket is larger than the value of *Size*, then the size of received data specified by *Size* is stored in *RecDat[].*
- The receive data is not read if the value of *Size* is 0.
- If the SktClose instruction closes the connection when there is no data in the receive buffer, an error end occurs even if a timeout has not occurred.
- You can execute a maximum of 32 of the following instructions at the same time: SktUDPCreate, SktUDPRcv, SktUDPSend, SktTCPAccept, SktTCPConnect, SktTCPRcv, SktTCPSend, SktGetTCP-Status, SktClose, SktClearBuf, and SktSetOption.
- An error occurs in the following cases. *Error* will change to TRUE.
  - There is a setting error for the local IP address.
  - The value of a member of *Socket* is outside of the valid range.
  - Data reception is in progress for the socket specified with *Socket.*
  - The socket specified with *Socket* is not connected.
  - The handle specified by *Socket.Handle* does not exist.
  - Data was not received before the time that is specified with *TimeOut* expired.
  - The socket was closed with the SktClose instruction.

Sample Programming

Refer to the sample programming that is provided for the SktTCPConnect instruction (page 2-1070).

SktTCPSend

The SktTCPSend instruction sends data from a TCP port for the built-in EtherNet/IP.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
SktTCPSend	TCP Socket Send	FB		SktTCPSend_instance(Execute, Socket, SendDat, Size, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Socket	Socket	Input	Socket	---	---	---
SendDat[] (array)	Send data		Send data	Depends on data type.		
Size	Send data size		Send data size	0 to 2000	Bytes	1

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Socket	Refer to *Function* for details on the structure _sSOCKET.																			
SendDat[] (array)		OK																		
Size							OK													

Function

The SktTCPSend instruction sends send data *SendDat[]* from the socket that is specified with *Socket*. The number of bytes to send is specified with *Size*.

The data type of *Socket* is structure _sSOCKET. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Socket	Socket	Socket	_sSOCKET	---	---	---
Handle	Handle	Handle for data communications	UDINT	Depends on data type.	---	0
SrcAdr*	Local address	Local IP address and port number	_sSOCKET_ ADDRESS	---	---	---
PortNo*	Port number	Port number	UINT	1 to 65535	---	0
IpAdr*	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		‘’
DstAdr*	Destination address	Destination IP address and port number	_sSOCKET_ ADDRESS	---	---	---
PortNo*	Port number	Port number	UINT	1 to 65535	---	0
IpAdr*	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		‘’

* These members are not used for this instruction.

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta*1	Online	BOOL	This variable indicates when built-in EtherNet/IP port communications can be used. TRUE: Communications are possible. FALSE: Communications are not possible.
_EIP1_EtnOnlineSta*2 _EIP2_EtnOnlineSta*3			
_EIPIn1_EtnOnlineSta*4			

*1 Use this variable name for an NJ-series CPU Unit.

*2 Use this variable name for port 1 on an NX-series CPU Unit, or for an NY-series Controller.

*3 Use this variable name for port 2 on an NX-series CPU Unit.

*4 Use this variable name for the internal communication port on an NY-series Controller.

Additional Information

Refer to the *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User’s Manual* (Cat. No. W506) or *NY-series Industrial Panel PC / Industrial Box PC Built-in EtherNet/IP Port User’s Manual* (Cat. No. W563) for details on socket services.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the built-in EtherNet/IP on NJ/NX-series CPU Units and NY-series Controllers.
- Up to 2,000 bytes of data can be sent with one instruction. A maximum of 2,000 bytes is sent even if the *SendDat[]* array is larger than 2,000 bytes.
- Data is not sent if the value of *Size* is 0.

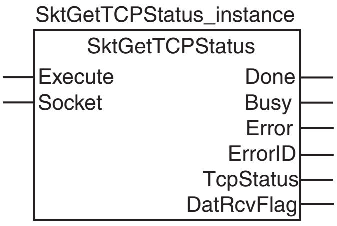
- You can execute a maximum of 32 of the following instructions at the same time: SktUDPCreate, SktUDPRcv, SktUDPSend, SktTCPAccept, SktTCPConnect, SktTCPRcv, SktTCPSend, SktGetTCP-Status, SktClose, SktClearBuf, and SktSetOption.
- An error occurs in the following cases. *Error* will change to TRUE.
  - There is a setting error for the local IP address.
  - The value of a member of *Socket* is outside of the valid range.
  - Data transmission is in progress for the socket specified with *Socket.*
  - The socket specified with *Socket* is not connected.
  - The handle specified by *Socket.Handle* does not exist.

Sample Programming

Refer to the sample programming that is provided for the SktTCPConnect instruction (page 2-1070).

SktGetTCPStatus

The SktGetTCPStatus instruction reads the status of a TCP socket.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
SktGetTCP Status	Read TCP Socket Status	FB		SktGetTCPStatus_instance(Execute, Socket, Done, Busy, Error, ErrorID, TcpStatus, DatRcvFlag);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Socket	Socket	Input	Socket	---	---	---
TcpStatus	TCP connection status	Output	TCP connection status	*	---	---
DatRcvFlag	Data Received Flag	Output	TRUE: Data is received. FALSE: Data is not received.	Depends on data type.	---	---

* _CLOSED, _LISTEN, _SYN_SENT, _SYN_RECEIVED, _ESTABLISHED, _CLOSE_WAIT, _FIN_WAIT1, _CLOSING, _LAST_ACK, _FIN_WAIT2, or _TIME_WAIT

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Socket	Refer to *Function* for details on the structure _sSOCKET.																			
TcpStatus	Refer to *Function* for the enumerators of the enumerated type _eCONNECTION_STATE.																			
DatRcvFlag	OK																			

Function

The SktGetTCPStatus instruction gets the TCP connection status *TcpStatus* of the socket that is specified with *Socket*. If there is receive data in the receive buffer, the value of data received flag *DatRecvFlag* changes to TRUE. Storage of the data to *TcpStatus* and *DatRcvFlag* is completed when the instruction is completed normally (i.e., when the value of *Done* changes to TRUE).

The data type of *Socket* is structure _sSOCKET. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Socket	Socket	Socket	_sSOCKET	---	---	---
Handle	Handle	Handle for data communications	UDINT	Depends on data type.	---	0
SrcAdr*	Local address	Local IP address and port number	_sSOCKET_ADDRESS	---	---	---
PortNo*	Port number	Port number	UINT	1 to 65535	---	0
IpAdr*	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		''
DstAdr*	Destination address	Destination IP address and port number	_sSOCKET_ADDRESS	---	---	---
PortNo*	Port number	Port number	UINT	1 to 65535	---	0
IpAdr*	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		''

* These members are not used for this instruction.

The data type of *TcpStatus* is enumerated type _eCONNECTION_STATE. The meanings of the enumerators of enumerated type _eCONNECTION_STATE are as follows:

Enumerators	Meaning
_CLOSED	CLOSED status
_LISTEN	LISTEN status
_SYN_SENT	SYN SENT status
_SYN_RECEIVED	SYN RECEIVED status
_ESTABLISHED	ESTABLISHED status
_CLOSE_WAIT	CLOSE WAIT status
_FIN_WAIT1	FIN WAIT1 status
_CLOSING	CLOSING status
_LAST_ACK	LAST ACK status
_FIN_WAIT2	FIN WAIT2 status
_TIME_WAIT	TIME WAIT status

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta*1	Online	BOOL	This variable indicates when built-in EtherNet/IP port communications can be used. TRUE: Communications are possible. FALSE: Communications are not possible.
_EIP1_EtnOnlineSta*2			
_EIP2_EtnOnlineSta*3			
_EIPIn1_EtnOnlineSta*4			

*1 Use this variable name for an NJ-series CPU Unit.

*2 Use this variable name for port 1 on an NX-series CPU Unit, or for an NY-series Controller.

*3 Use this variable name for port 2 on an NX-series CPU Unit.

*4 Use this variable name for the internal communication port on an NY-series Controller.

Additional Information

Refer to the *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User's Manual* (Cat. No. W506) or *NY-series Industrial Panel PC / Industrial Box PC Built-in EtherNet/IP Port User's Manual* (Cat. No. W563) for details on socket services.

Precautions for Correct Use

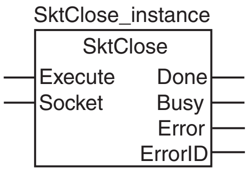
- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the built-in EtherNet/IP on NJ/NX-series CPU Units and NY-series Controllers.
- You can execute a maximum of 32 of the following instructions at the same time: SktUDPCreate, SktUDPRcv, SktUDPSend, SktTCPAccept, SktTCPConnect, SktTCPRcv, SktTCPSend, SktGetTCP-Status, SktClose, SktClearBuf, and SktSetOption.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The value of a member of *Socket* is outside of the valid range.
  - The handle specified by *Socket.Handle* does not exist.

Sample Programming

Refer to the sample programming that is provided for the SktTCPConnect instruction (page 2-1070).

SktClose

The SktClose instruction closes the specified TCP or UDP socket for the built-in EtherNet/IP.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
SktClose	Close TCP/UDP Socket	FB		SktClose_instance(Execute, Socket, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Socket	Socket	Input	Socket	---	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Socket	Refer to *Function* for details on the structure _sSOCKET.																			

Function

The SktClose instruction closes the socket that is specified with *Socket*. If a TCP socket is specified, the socket is disconnected before it is closed. If the socket handle *Socket.Handle* is 0, all TCP and UDP ports that currently use the socket service are closed. Close processing for the TCPUDP sockets is completed when the instruction is completed normally (i.e., when the value of *Done* changes to TRUE).

The data type of *Socket* is structure _sSOCKET. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Socket	Socket	Socket	_sSOCKET	---	---	---
Handle	Handle	Handle of the connection to close. 0: Closes all TCP connections that currently use the socket service.	UDINT	Depends on data type.	---	0
SrcAdr*	Local address	Local IP address and port number	_sSOCKET_ ADDRESS	---	---	---
PortNo*	Port number	Port number	UINT	1 to 65535	---	0
IpAdr*	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		‘’
DstAdr*	Destination address	Destination IP address and port number	_sSOCKET_ ADDRESS	---	---	---
PortNo*	Port number	Port number	UINT	1 to 65535	---	0
IpAdr*	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		‘’

* These members are not used for this instruction.

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta*1	Online	BOOL	This variable indicates when built-in EtherNet/IP port communications can be used. TRUE: Communications are possible. FALSE: Communications are not possible.
_EIP1_EtnOnlineSta*2			
_EIP2_EtnOnlineSta*3			
_EIPIn1_EtnOnlineSta*4			

*1 Use this variable name for an NJ-series CPU Unit.

*2 Use this variable name for port 1 on an NX-series CPU Unit, or for an NY-series Controller.

*3 Use this variable name for port 2 on an NX-series CPU Unit.

*4 Use this variable name for the internal communication port on an NY-series Controller.

Additional Information

Refer to the *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User’s Manual* (Cat. No. W506) or *NY-series Industrial Panel PC / Industrial Box PC Built-in EtherNet/IP Port User’s Manual* (Cat. No. W563) for details on socket services.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the built-in EtherNet/IP on NJ/NX-series CPU Units and NY-series Controllers.

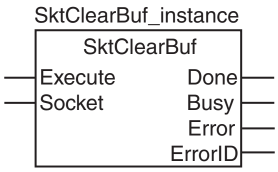
- If the SktUDPRcv or SktTCPRcv instruction is executed and then the SktClose instruction is executed while the socket for the specified handle is on standby to received data, the standby status is canceled.
- If more than one connection is open for the same local port number, only the connection for the specified socket is closed.
- If the value of the socket handle *Socket.Handle* is 0, all connections that are on standby for the SktTCPAccept instruction are canceled.
- You can execute a maximum of 32 of the following instructions at the same time: SktUDPCreate, SktUDPRcv, SktUDPSend, SktTCPAccept, SktTCPConnect, SktTCPRcv, SktTCPSend, SktGetTCPStatus, SktClose, SktClearBuf, and SktSetOption.
- An error occurs in the following cases. *Error* will change to TRUE.
  - There is a setting error for the local IP address.
  - The value of a member of *Socket* is outside of the valid range.
  - The handle specified by *Socket.Handle* does not exist.

Sample Programming

Refer to the sample programming for the following instructions: SktUDPCreate (page 2-1053) and SktTCPConnect (page 2-1070).

SktClearBuf

The SktClearBuf instruction clears the receive buffer for the specified TCP or UDP socket for the built-in EtherNet/IP.

Instruction	Name	FB/FUN	Graphic expression	ST expression
SktClearBuf	Clear TCP/UDP Socket Receive Buffer	FB		SktClearBuf_instance(Execute, Socket, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Socket	Socket	Input	Socket	---	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Socket	Refer to *Function* for details on the structure _sSOCKET.																			

Function

The SktClearBuf instruction clears the receive buffer for the socket that is specified with *Socket*. Clear processing of the receive buffer is completed when the instruction is completed normally (i.e., when the value of *Done* changes to TRUE).

The data type of *Socket* is structure _sSOCKET. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Socket	Socket	Socket	_sSOCKET	---	---	---
Handle	Handle	The handle of the socket to clear the receive buffer	UDINT	Depends on data type.	---	0
SrcAdr*	Local address	Local IP address and port number	_sSOCKET_ ADDRESS	---	---	---
PortNo*	Port number	Port number	UINT	1 to 65535	---	0
IpAdr*	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		‘’
DstAdr*	Destination address	Destination IP address and port number	_sSOCKET_ ADDRESS	---	---	---
PortNo*	Port number	Port number	UINT	1 to 65535	---	0
IpAdr*	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		‘’

* These members are not used for this instruction.

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta*1	Online	BOOL	This variable indicates when built-in EtherNet/IP port communications can be used. TRUE: Communications are possible. FALSE: Communications are not possible.
_EIP1_EtnOnlineSta*2			
_EIP2_EtnOnlineSta*3			
_EIPIn1_EtnOnlineSta*4			

*1 Use this variable name for an NJ-series CPU Unit.

*2 Use this variable name for port 1 on an NX-series CPU Unit, or for an NY-series Controller.

*3 Use this variable name for port 2 on an NX-series CPU Unit.

*4 Use this variable name for the internal communication port on an NY-series Controller.

Additional Information

Refer to the *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User’s Manual* (Cat. No. W506) or *NY-series Industrial Panel PC / Industrial Box PC Built-in EtherNet/IP Port User’s Manual* (Cat. No. W563) for details on socket services.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the built-in EtherNet/IP on NJ/NX-series CPU Units and NY-series Controllers.
- You can execute a maximum of 32 of the following instructions at the same time: SktUDPCreate, SktUDPRcv, SktUDPSend, SktTCPAccept, SktTCPConnect, SktTCPRcv, SktTCPSend, SktGetTCP-Status, SktClose, SktClearBuf, and SktSetOption.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The value of a member of *Socket* is outside of the valid range.

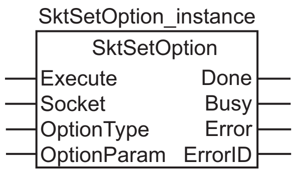
- The socket that is specified by *Socket* does not exist.
- The handle specified by *Socket.Handle* does not exist.

Sample Programming

Refer to the sample programming that is provided for the SktTCPConnect instruction (page 2-1070).

SktSetOption

The SktSetOption instruction sets the option for TCP socket specified for the built-in EtherNet/IP.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
SktSetOption	Set TCP Socket Option	FB		SktSetOption_instance(Execute, Socket, OptionType, OptionParam, Done, Busy, Error, ErrorID);

Version Information

A CPU Unit with unit version 1.12 or later and Sysmac Studio version 1.16 or higher are required to use this instruction.

For an NX1P2 CPU Unit, a CPU Unit with unit version 1.14 or later and Sysmac Studio 1.18 or higher are required to use this instruction.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Socket	Socket	Input	Socket	---	---	---
OptionType	Option type		Type of socket option	---	---	---
OptionParam	Option parameter		Setting parameters according to the specified socket option	---	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Socket	Refer to *Function* for details on the structure _sSOCKET.																			
OptionType	Refer to *Function* for the enumerators of the enumerated type _eSKT_OPTION_TYPE.																			
OptionParam	OK[*1]																			

*1 A constant (literal) cannot be input. Specify a variable.

Function

The SktSetOption instruction sets the socket option for the socket specified with *Socket.*

Done changes to TRUE when processing of the instruction is completed normally.

The socket option setting is completed when processing of the instruction is completed normally.

The data type of Socket is structure _sSOCKET. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Socket	Socket	Socket	_sSOCKET	---	---	---
Handle	Handle	The handle of the socket to clear the receive buffer	UDINT	Depends on data type.	---	0
SrcAdr(*)	Local address	Local IP address and port number	_sSOCKET_ADDRESS	---	---	---
PortNo(*)	Port number	Port number	UINT	1 to 65,535	---	0
IpAdr(*)	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		‘’
DstAdr(*)	Destination address	Destination IP address and port number	_sSOCKET_ADDRESS	---	---	---
PortNo(*)	Port number	Port number	UINT	1 to 65,535	---	0
IpAdr(*)	IP address	IP address or host name. A DNS or Hosts setting is required to use a host name.	STRING	Depends on data type.		‘’

* These members are not used for this instruction.

The following table shows the value of *OptionType* that you can specify and the data type of *OptionParam* that you can select for the specified *OptionType*. Also, the default operation when this instruction is not used is given by the default value below.

OptionType		OptionParam		
Enumerator	Meaning	Selectable data type	Meaning of value	Default
_TCP_NODELAY	Specifies the TCP_NODELAY option. It can be used only for TCP socket.	BOOL	TRUE[*1] : TCP_NODELAY option enabled FALSE: TCP_NODELAY option disabled	FALSE

*1 When it is set to TRUE, the Nagle algorithm is disabled. With this setting, even small data is not transmitted collectively.

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta *1	Online	BOOL	This variable indicates when built-in EtherNet/IP port communications can be used. TRUE: Communications are possible. FALSE: Communications are not possible.
_EIP1_EtnOnlineSta *2 _EIP2_EtnOnlineSta *3			
_EIPIn1_EtnOnlineSta *4			

*1 Use this variable name for an NJ-series CPU Unit.
*2 Use this variable name for port 1 on an NX-series CPU Unit, or for an NY-series Controller.
*3 Use this variable name for port 2 on an NX-series CPU Unit.
*4 Use this variable name for the internal communication port on an NY-series Controller.

Additional Information

Refer to the *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User's Manual* (Cat. No. W506) or the *NY-series Industrial Panel PC / Industrial Box PC Built-in EtherNet/IP Port User's Manual* (Cat. No. W563) for details on the socket service functions.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction can be used only for the built-in EtherNet/IP on the NJ/NX-series CPU Units and on the NY-series Controllers.
- You can use this instruction after the socket handle is opened by the SktTCPAccept, or SktTCPConnect instruction and before data transmission is started by SktTCPRcv, SktTCPSend, or SktClearBuf instruction. An error will occur if you execute this instruction after data transmission is started.
- You must set the socket option for each handle specified with *Socket*. The socket option that was set is enabled while the handle is open. After closing the handle with the SktClose instruction, execute the SktTCPAccept and SktTCPConnect instructions again to open the handle, and then execute this instruction to set the socket option.
- You can execute a maximum of 32 of the following instructions at the same time: SktUDPCreate, SktUDPRcv, SktUDPSend, SktTCPAccept, SktTCPConnect, SktTCPRcv, SktTCPSend, SktGetTCPStatus, SktClose, SktClearBuf, and SktSetOption.
- An error will occur in the following cases. *Error* will change to TRUE.
  - The value of a member of *Socket* is outside of the valid range.
  - The data type specified for *OptionParam* is not supported by *OptionType*.
  - The specified handle socket already started transmission.
  - The specified socket type is not supported by the handle type. Such a case includes when the TCP_NODELAY is executed for UDP socket.
  - The handle specified by *Socket.Handle* does not exist.

Sample Programming

ST

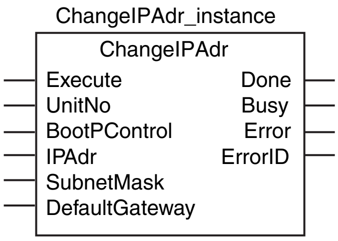
Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	FALSE	Execution condition
	DoTCP	BOOL	FALSE	Processing
	Stage	INT	0	State transition
	WkSocket	_sSOCKET	(Handle:=0, SrcAdr:=(PortNo:=0,IpAdr:=''), DstAdr:=(PortNo:=0,IpAdr:=''))	Socket
	SendSocketDat	ARRAY[0..1999] OF BYTE		Send data
	Nodelay	BOOL	TRUE	*NoDelay* setting
	SktTCPConnect_instance	SktTCPConnect		
	SktSetOption_instance	SktSetOption		
	SktTCPSend_instance	SktTCPSend		
	SktClose_instance	SktClose		

```
// Start sequence when Trigger changes to TRUE.
IF ((Trigger=TRUE) AND (DoTCP=FALSE) AND (_EIP_EtnOnlineSta=TRUE)) THEN
        DoTCP:=TRUE;
        Nodelay:=TRUE;
        Stage:=INT#1;
        SktTCPConnect_instance(Execute:=FALSE);// Initialize instance.
        SktSetOption_instance( // Initialize instance.
              Execute:=FALSE,
              OptionType:=_TCP_NODELAY,
              OptionParam:= Nodelay);
        SktSetOption_instance(Execute:=FALSE);// Initialize instance.
        SktTCPSend_instance(// Initialize instance.
              Execute:=FALSE,
              SendDat:=SendSocketDat[0]);// Dummy
        SktClose_instance(Execute:=FALSE);// Initialize instance.
END_IF;

IF (DoTCP=TRUE) THEN
        CASE Stage OF
        1 :// Request a connection.
              SktTCPConnect_instance(
                  Execute:=TRUE,
                  SrcTcpPort:=UINT#0,// Local UDP port number: Automatically
                  assigned.
                  DstAdr:='192.168.250.2',// Remote IP address
                  DstTcpPort:=UINT#6000,// Destination TCP port number
                  Socket =>WkSocket);// Socket
              IF (SktTCPConnect_instance.Done=TRUE) THEN
                  Stage:=INT#2;// Normal end
              ELSIF (SktTCPConnect_instance.Error=TRUE) THEN
                  Stage:= INT#10; // Error end
```

```
            END_IF;

        2 :// Set Socket Option
             SktSetOption_instance(
                  Execute:=TRUE,
                  Socket:=WkSocket);// Socket
                  OptionType:=_TCP_NODELAY, // Option type
                  OptionParam:= Nodelay); // NODELAY enabled
             IF (SktSetOption_instance.Done=TRUE) THEN
                  Stage:=INT#3;// Normal end
             ELSIF (SktSetOption_instance.Error=TRUE) THEN
                  Stage:=INT#20; // Error end
                  END_IF;

        3 :// Send request
             SktTCPSend_instance(
                  Execute:=TRUE,
                  Socket:=WkSocket);// Socket
                  SendDat:=SendSocketDat[0],// Send data
                  Size:=UINT#2000);// Send data size
             IF (SktTCPSend_instance.Done=TRUE) THEN
                  Stage:=INT#4;// Normal end
             ELSIF (SktTCPSend_instance.Error=TRUE) THEN
                  Stage:= INT#30; // Error end
             END_IF;

        4 :// Request closing data.
             SktClose_instance(
                     Execute:=TRUE,
                     Socket:=WkSocket);// Socket
             IF (SktClose_instance.Done=TRUE) THEN
                     Stage:=INT#0;// Normal end
             ELSIF (SktClose_instance.Error=TRUE) THEN
                     Stage:= INT#40; // Error end
             END_IF;

        0 :// Normal end
             DoTCP:=FALSE;
             Trigger:=FALSE;

        ELSE// Interrupted by error.
             DoTCP:=FALSE;
             Trigger:=FALSE;
     END_CASE;
END_IF;
```

ChangeIPAdr

The ChangeIPAdr instruction changes the IP address of the built-in EtherNet/IP port on a CPU Unit or the IP address of an EtherNet/IP Unit.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
ChangeIPAdr	Change IP Address	FB		ChangeIPAdr_instance(Execute, UnitNo, BootPControl, IPAdr, SubnetMask, DefaultGateway, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
UnitNo	Unit number	Input	Unit number for which to change the IP address	_CBU_CPU *1, _CBU_CPU_Port1 *2, _CBU_CPU_Port2 *3, _CBU_CPU_InPort1 *4, or _CBU_No00 to _CBU_No15 *5	---	_CBU_No00
BootP Control	IP address assignment method and setting timing		Method to obtain the IP address and the setting timing	0 to 3 *6		0
IPAdr[] (array)*7	IP address		IP address	*8		---
Subnet Mask[] (array)*7	Subnet mask		Subnet mask			
Default Gateway[] (array)*7	Default gateway		Default gateway			

*1 Specification is possible for an NJ-series CPU Unit.

*2 Specification is possible for port 1 on an NX-series CPU Unit or an NY-series Controller. You can specify _CBU_CPU instead of _CBU_CPU_Port1.

*3 Specification is possible for port 2 on an NX-series CPU Unit.

*4 Specification is possible for an NY-series Controller.

*5 Specification is possible for an NJ-series CPU Unit.

*6 The range is 0 to 2 for port 1 on an NX-series CPU Unit, for NJ-series CPU Unit, and for an NY-series Controller. The range is 0 to 3 for port 2 on an NX-series CPU Unit. The range is 0 for the internal port on an NY-series Controller.

*7 This is a 4-element array with element numbers 0 to 3.

*8 The valid range depends on whether you specify the built-in EtherNet/IP port or an EtherNet/IP Unit for *UnitNo* (Unit number). Refer to *Function*, below, for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
UnitNo	Refer to *Function* for the enumerators of the enumerated type _eUnitNo.																			
BootP Control							OK													
IPAdr[] (array)		OK																		
	Specify an array.																			
Subnet Mask[] (array)		OK																		
	Specify an array.																			
BootP Control		OK																		
Default Gateway[] (array)	Specify an array.																			

Function

The ChangeIPAdr instruction changes the IP address of the built-in EtherNet/IP port, internal communication port, or EtherNet/IP Unit that is specified with *UnitNo* (Unit number) according to IP address assignment method and setting timing *BootPControl*.

If you specify the built-in EtherNet/IP port with *UnitNo*, the port goes to link OFF status when execution of the instruction ends and then goes to link ON status with the new IP address.

If the internal communication port is specified with *UnitNo*, the *_EIPIn1_EtnOnlineSta* system-defined variable changes to FALSE when execution of the instruction ends. Communications with the new IP address is enabled when the variable changes to TRUE. If you specify an EtherNet/IP Unit with *UnitNo*, the EtherNet/IP Unit is restarted when execution of the instruction ends. Communications with the new IP address is enabled when restarting the Unit ends.

You can use this instruction to change the IP address of the built-in EtherNet/IP port, internal communication port, or an EtherNet/IP Unit from an HMI.

The data type of *UnitNo* is enumerated type _eUnitNo. The meanings of the enumerators are as follows:

Enumerator	Meaning
_CBU_CPU*1	Built-in EtherNet/IP port
_CBU_CPU_Port1*2	Built-in EtherNet/IP communications port 1
_CBU_CPU_Port2*3	Built-in EtherNet/IP communications port 2
_CBU_CPU_InPort1*4	Internal port 1
_CBU_No00 to _CBU_-No15*5	Unit number 00 to 15 of the EtherNet/IP Unit

*1 Specification is possible for an NJ-series CPU Unit.
*2 Specification is possible for an NX-series CPU Unit or an NY-series Controller. You can specify _CBU_CPU instead of _CBU_CPU_Port1.
*3 Specification is possible for an NX-series CPU Unit. You cannot use it for CPU Units without communications port 2.
*4 Specification is possible for an NY-series Controller.
*5 Specification is possible for an NJ-series CPU Unit.

The value of *BootPControl* determines how to obtain the new IP address and when to set it, as described in the following table.

For *BootPControl*, you can specify a value in the range of 0 to 2 for port 1 on an NX-series CPU Unit, for an NJ-series CPU Unit, and for an NY-series Controller. The range is 0 to 3 for port 2 on an NX-series CPU Unit. You can specify only 0 for internal communication port on an NY-series Controller.

Value of *BootPControl*	Method to obtain the IP address	When to change the IP address
0	The IP address is obtained from IP address *IPAdr[]*, subnet mask *SubnetMask[]*, and default gateway *DefaultGateway[]*.	The IP address is set only once each time the instruction is executed (fixed setting).
1	The IP address is obtained from the BOOTP server.	The IP address is set once when the instruction is executed and then once each time the power supply to the Controller is turned ON.
2	The IP address is obtained from the BOOTP server.	The IP address is set only once each time the instruction is executed (fixed setting).
3	The port is set to an unused port. Any existing IP address is deleted.	The IP address is set only once each time the instruction is executed (fixed setting).

Set the IP address, subnet mask, and default gateway in order in elements 0 to 3 of *IPAdr[]*, *SubnetMask[]*, and *DefaultGateway[]*. For example, if the new IP address is 130.58.17.32, set IPAdr[0] to BYTE#16#82, *IPAdr[1]* to BYTE#16#3A, *IPAdr[2]* to BYTE#16#11 and *IPAdr[3]* to BYTE#16#20.

The valid ranges of *IPAdr[]*, *SubnetMask[]*, and *DefaultGateway[]* depend on whether you specify the built-in EtherNet/IP port or an EtherNet/IP Unit for *UnitNo*, as shown below. The valid ranges of the values are valid only when the value of *BootPControl* is set to 0.

Setting of *UnitNo*	Input variable	Valid range
Built-in EtherNet/IP port Internal communication port	IPAdr[] (array)	The following IP addresses cannot be used. All other IP addresses are valid. • IP addresses that start with 127, 0, or 255 • IP addresses with a host ID for which all bits are 0’s or for which all bits are 1’s • Class D IP addresses (224.0.0.0 to 239.255.255.255) • Class E IP addresses (240.0.0.0 to 255.255.255.255) • IP addresses that are reserved for AutoIP*1 (169.254.0.0 to 169.254.255.255) • IP addresses of USB ports (192.168.255.0 to 192.168.255.255)*2
	SubnetMask[] (array)	192.0.0.0 to 255.255.255.252
	DefaultGateway[] (array)	The following IP addresses cannot be used. All other IP addresses are valid. • IP addresses that start with 127, 0, or 255 • There is only one address for which all bits are 1’s • Class D IP addresses (224.0.0.0 to 239.255.255.255) • Class E IP addresses (240.0.0.0 to 255.255.255.255) • IP addresses that are reserved for AutoIP*1 (169.254.0.0 to 169.254.255.255) • IP addresses of USB ports (192.168.255.0 to 192.168.255.255)*2

Setting of ***UnitNo***	Input variable	Valid range
EtherNet/IP Unit	IPAdr[] (array)	The following IP addresses cannot be used. All other IP addresses are valid. • IP addresses that start with 127 • Class D IP addresses (224.0.0.0 to 239.255.255.255) • Class E IP addresses (240.0.0.0 to 255.255.255.255)
	SubnetMask[] (array)	• 0.0.0.0 • 192.0.0.0 to 255.255.255.252
	DefaultGateway[] (array)	The following IP addresses cannot be used. All other IP addresses are valid. • IP addresses that start with 127 • Class D IP addresses (224.0.0.0 to 239.255.255.255) • Class E IP addresses (240.0.0.0 to 255.255.255.255)

*1 AutoIP is an automatic IP address assignment feature of Windows 98 and later operating systems.

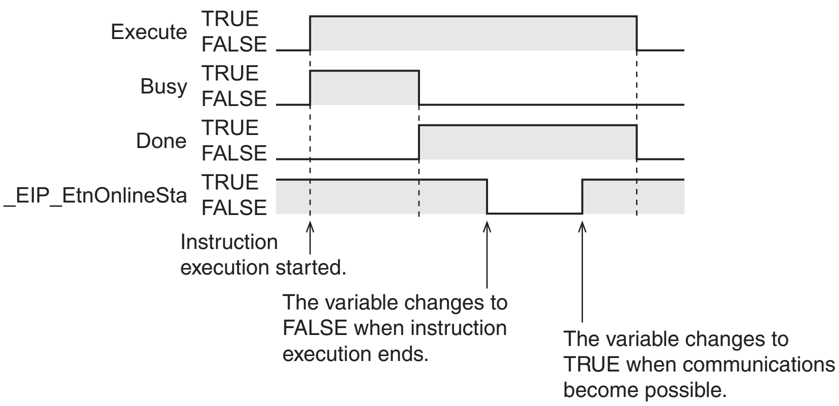
*2 NX1P2 CPU Units and NY-series Controllers have no USB port.

The values of *IPAdr[]*, *SubnetMask[]*, and *DefaultGateway[]* are ignored when the value of *BootPControl* is 1 or 2. Therefore, the values of *IPAdr[]*, *SubnetMask[]*, and *DefaultGateway[]* can be outside of the valid ranges.

If you specify the built-in EtherNet/IP port for *UnitNo*, you can use the *_EIP_EtnOnlineSta*, *_EIP1_EtnOnlineSta, _EIP2_EtnOnlineSta*, and *_EIPIn1_EtnOnlineSta* system-defined variables to see if communications are possible.

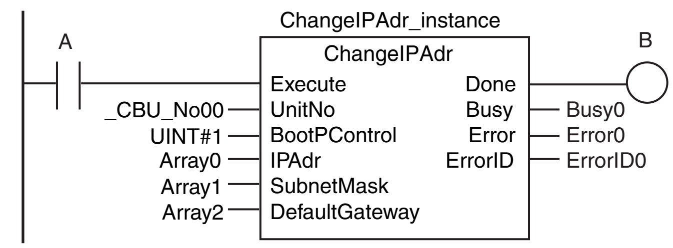
Here, *_EIP_EtnOnlineSta* is used as an example, but this information also applies to *_EIP1_EtnOnlineSta, _EIP2_EtnOnlineSta* and *_EIPIn1_EtnOnlineSta.*

When *Busy* changes to FALSE, *_EIP_EtnOnlineSta* changes to FALSE. When communications using the new IP address are enabled, *_EIP_EtnOnlineSta* changes to TRUE.



The following example shows how to change the IP address of the EtherNet/IP Unit with unit number 00 to the IP address that is obtained from the BOOTP server each time the instruction is executed. If A *(Execute)* is changed to TRUE from an HMI or other device, the IP address is changed to the IP address that is obtained from the BOOTP server. Then, each time the power supply is turned ON, the IP address is changed to the IP address that is obtained from the BOOTP server.

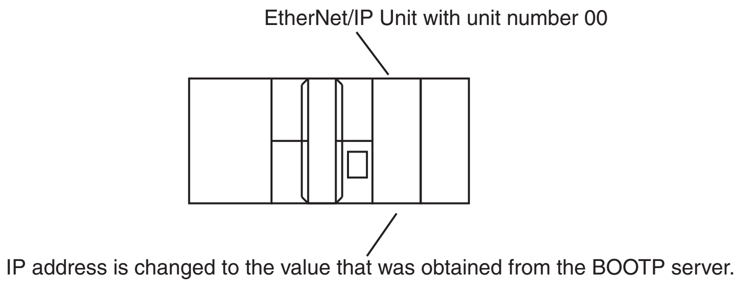
LD



ST

```
ChangeIPAdr_instance(A,_CBU_No00,UINT#1,Array0,Array1,Array2,B,Busy0,Error0,ErrorID0);
```

The IP address that was obtained from the BOOTP server is set for the EtherNet/IP Unit with a unit number of 00.
Then, each time the power supply is turned ON, the IP address is reset to the IP address that is obtained from the BOOTP server.



Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta*1 _EIP1_EtnOnlineSta*2 _EIP2_EtnOnlineSta*3 _EIPIn1_EtnOnlineSta*4	Online	BOOL	This variable indicates when built-in EtherNet/IP port communications can be used. TRUE: Communications are possible. FALSE: Communications are not possible.

*1 Use this variable name for an NJ-series CPU Unit.

*2 Use this variable name for port 1 on an NX-series CPU Unit, or for an NY-series Controller.

*3 Use this variable name for port 2 on an NX-series CPU Unit.

*4 Use this variable name for the internal communication port on an NY-series Controller.

Additional Information

- If you specify the built-in EtherNet/IP port with *UnitNo*, the following events are recorded in the event log when the instruction is executed.
  - Link OFF Detected
  - IP Address Fixed
- If you specify the internal communication port with *UnitNo*, the following events are recorded in the event log when the instruction is executed.
  - IP Address Fixed
- You can change the IP address with this instruction even if the CPU Unit is write protected.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- If you specify the built-in EtherNet/IP port with *UnitNo*, communications with the built-in EtherNet/IP port will be disabled temporarily when execution of the instruction ends. Confirm that the system will not be adversely affected even if the built-in EtherNet/IP port goes to link OFF status.
- If you specify an EtherNet/IP Unit with *UnitNo*, the EtherNet/IP Unit is restarted when execution of the instruction ends. Confirm that the system will not be adversely affected even if the EtherNet/IP Unit is restarted.
- You cannot use this instruction in an event task. A compiling error will occur.
- *Error* is TRUE if an error occurred. The meanings of the values of *ErrorID* are given in the following table.

Value of *ErrorID*	Error name	Description
16#0400	Input Value Out of Range	The value of an input variable is outside of the valid range.
16#2400	No Execution Right	The instruction was executed when changing the status was not possible. • While changing the settings was already in progress • While restarting the built-in EtherNet/IP port was in progress • While downloading tag data link settings from the Network Configurator was in progress
16#2402	Too Many Simultaneous Instruction Executions	Too many ChangeIPAdr, ChangeFTPAccount, and ChangeNTPServerAdr instructions were executed at the same time.
16#240D	IP Address Setting Invalid	The network address of the specified port is the same as the network address of another port.

Version Information

A CPU Unit with unit version 1.02 or later and Sysmac Studio version 1.03 or higher are required to use this instruction.

Sample Programming

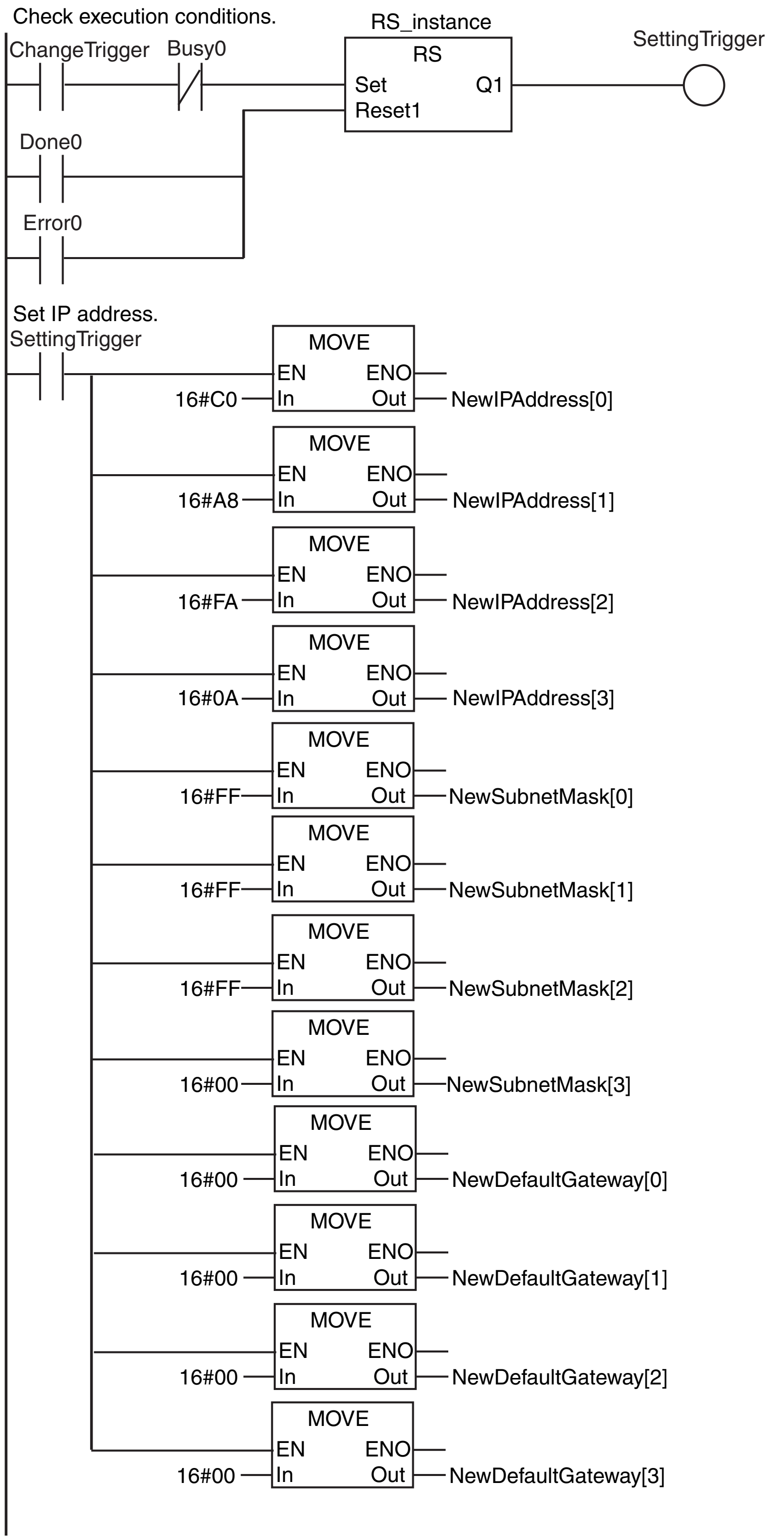
This sample changes the IP address of the built-in EtherNet/IP port to the following fixed IP address.

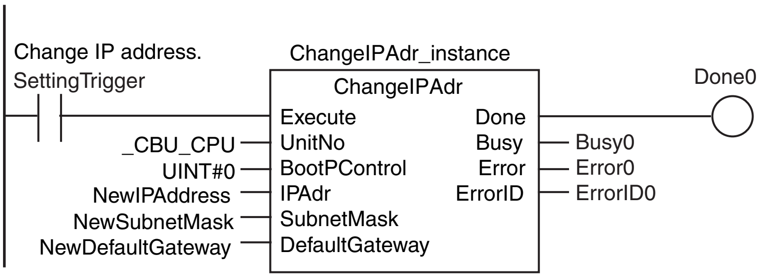
Item	Value
IP address	192.168.250.10
Subnet mask	255.255.255.0
Default gateway	0.0.0.0

LD

Variable	Data type	Initial value	Comment
ChangeTrigger	BOOL	False	Change Flag
SettingTrigger	BOOL	False	Changing IP Address Flag
Done0	BOOL	False	IP address changed
Error0	BOOL	False	Error in changing the IP address
Busy0	BOOL	False	Changing IP address
ErrorID0	WORD	16#0	Error ID for changing the IP address

Variable	Data type	Initial value	Comment
NewIPAddress	ARRAY[0..3] OF BYTE	[4(16#0)]	IP address
NewSubnetMask	ARRAY[0..3] OF BYTE	[4(16#0)]	Subnet mask
NewDefaultGateway	ARRAY[0..3] OF BYTE	[4(16#0)]	Default gateway
RS_instance	RS		
ChangeIPAdr_instance	ChangeIPAdr		





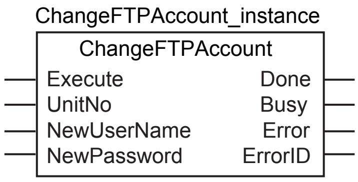
ST

Variable	Data type	Initial value	Comment
ChangeTrigger	BOOL	False	Change Flag
SettingTrigger	BOOL	False	Changing IP Address Flag
Done0	BOOL	False	IP address changed
Error0	BOOL	False	Error in changing the IP address
Busy0	BOOL	False	Changing IP address
ErrorID0	WORD	16#0	Error ID for changing the IP address
NewIPAddress	ARRAY[0..3] OF BYTE	[4(16#0)]	IP address
NewSubnetMask	ARRAY[0..3] OF BYTE	[4(16#0)]	Subnet mask
NewDefaultGateway	ARRAY[0..3] OF BYTE	[4(16#0)]	Default gateway
RS_instance	RS		
ChangeIPAdr_instance	ChangeIPAdr		

```
//Check execution conditions.
IF((ChangeTrigger=TRUE)AND(Busy0=FALSE))THEN
   SettingTrigger:= TRUE;
END_IF;

IF((Done0=TRUE)OR(Error0=TRUE))THEN
   SettingTrigger:= FALSE;
END_IF;

//Set IP address.
IF(SettingTrigger=TRUE)THEN
   NewIPAddress[0]      := 16#C0;
   NewIPAddress[1]      := 16#A8;
   NewIPAddress[2]      := 16#FA;
   NewIPAddress[3]      := 16#0A;
   NewSubnetMask[0]     := 16#FF;
   NewSubnetMask[1]     := 16#FF;
   NewSubnetMask[2]     := 16#FF;
   NewSubnetMask[3]     := 16#00;
   NewDefaultGateway[0]:= 16#00;
   NewDefaultGateway[1]:= 16#00;
   NewDefaultGateway[2]:= 16#00;
   NewDefaultGateway[3]:= 16#00;
END_IF;

//Change IP address.
ChangeIPAdr_instance(
          Execute        := SettingTrigger,
          UnitNo         := _CBU_CPU,
          BootPControl   := UINT#0,
          IPAdr          := NewIPAddress,
          SubnetMask     := NewSubnetMask,
          DefaultGateway := NewDefaultGateway,
          Done            =>Done0,
          Busy            =>Busy0,
          Error           =>Error0,
          ErrorID         =>ErrorID0);
```

ChangeFTPAccount

The ChangeFTPAccount instruction changes the FTP login name and password of the built-in EtherNet/IP port on a CPU Unit or those of an EtherNet/IP Unit.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
ChangeFTP Account	Change FTP Account	FB		ChangeFTPAccount_instance(Execute, UnitNo, NewUserName, NewPassword, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
UnitNo	Unit number	Input	Unit number for which to change the FTP login name and password	_CBU_CPU or _CBU_No00 to _CBU_No15*1	---	_CBU_No00
New UserName	Login name		Login name	1 to 12 single-byte alphanumeric characters (case sensitive)		---
New Password	Password		Password	*2		

*1 You can set *_CBU_No00* to *_CBU_No15* only for an NJ-series CPU Unit.

*2 The valid range depends on whether you specify the built-in EtherNet/IP port or an EtherNet/IP Unit for *UnitNo* (Unit number). Refer to Function, below, for details.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
UnitNo	Refer to *Function* for the enumerators of the enumerated type _eUnitNo.																			
New UserName																				OK
New Password																				OK

Function

The ChangeFTPAccount instruction changes the FTP login name and password of the built-in EtherNet/IP port or EtherNet/IP Unit that is specified with *UnitNo* (Unit number), to the values specified with FTP login name *NewUserName* and password *NewPassword*.

When *Execute* changes from FALSE to TRUE, the values of *NewUserName* and *NewPassword* are written as the FTP login name and password of the built-in EtherNet/IP port. The value of *Busy* remains TRUE during execution of the instruction, and the value of *Done* changes to TRUE when reception of the setting change request is completed. The settings are not applied yet when *Done* changes to TRUE.

If you specify an EtherNet/IP Unit with *UnitNo*, the EtherNet/IP Unit is restarted when execution of the instruction ends. The new login name and password are enabled when restarting the Unit ends.

You can use this instruction to change the FTP login name and password of the built-in EtherNet/IP port or an EtherNet/IP Unit from an HMI.

The data type of *UnitNo* is enumerated type _eUnitNo. The meanings of the enumerators are as follows:

Enumerator	Meaning
_CBU_CPU	Built-in EtherNet/IP port
_CBU_No00 to _CBU_-No15*1	Unit number 00 to 15 of the EtherNet/IP Unit

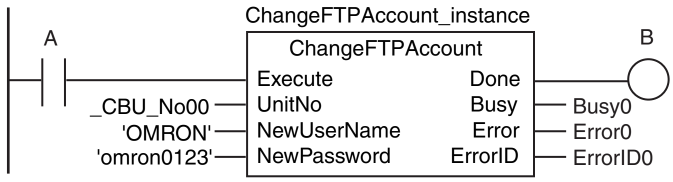
*1 This can be set only for an NJ-series CPU Unit.

The valid range of the value of *NewPassword* depends on whether you specify the built-in EtherNet/IP port or an EtherNet/IP Unit for *UnitNo* (Unit number), as given below.

Setting of *UnitNo*	Valid range
Built-in EtherNet/IP port	8 to 32 single-byte alphanumeric characters (case sensitive)
EtherNet/IP Unit	1 to 8 single-byte alphanumeric characters (case sensitive)

The following example shows how to change the FTP login name and password of the EtherNet/IP Unit with unit number 00. If A (Execute) is changed to TRUE from an HMI or other device, the FTP login name is changed to ‘OMRON’ and the password is changed to ‘omron0123’.

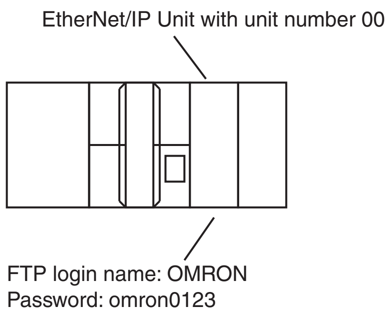
LD

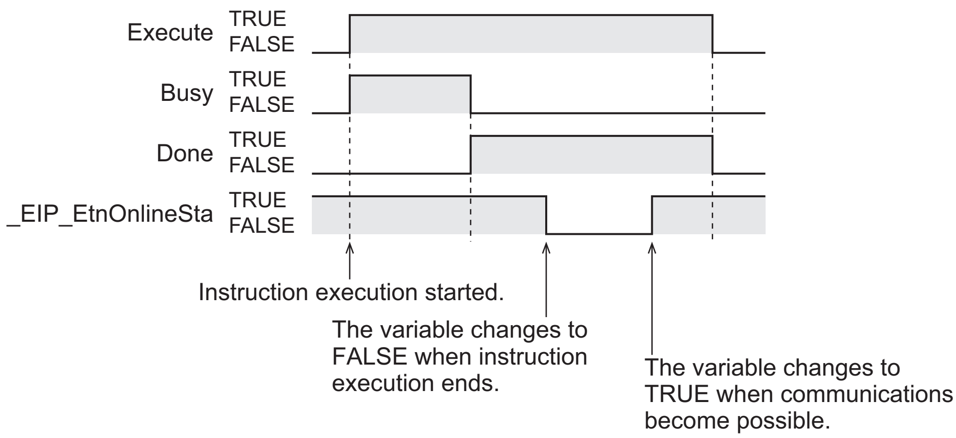


ST

```
ChangeFTPAccount_instance(A,_CBU_No00,'OMRON','omron0123',B,Busy0,Error0,ErrorID0);
```

The FTP login name is changed to ‘OMRON’ and the password is changed to ‘omron0123’ for the EtherNet/IP Unit with unit number 00.





Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta*1	Online	BOOL	This variable indicates when built-in EtherNet/IP port communications can be used. TRUE: Communications are possible. FALSE: Communications are not possible.
_EIP1_EtnOnlineSta*2 _EIP2_EtnOnlineSta*3			
_EIPIn1_EtnOnlineSta*4			

*1 Use this variable name for an NJ-series CPU Unit.

*2 Use this variable name for port 1 on an NX-series CPU Unit, or for an NY-series Controller.

*3 Use this variable name for port 2 on an NX-series CPU Unit.

*4 Use this variable name for the internal communication port on an NY-series Controller.

Additional Information

- You can change the FTP login name and password with this instruction even if the CPU Unit is write protected.
- Even if you change the FTP login name and password with this instruction during a file transfer, the file transfer will continue.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- You cannot use this instruction in an event task. A compiling error will occur.
- *Error* is TRUE if an error occurred. The meanings of the values of *ErrorID* are given in the following table.

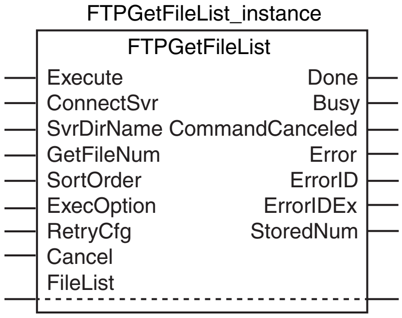
Value of *ErrorID*	Error name	Description
16#0400	Input Value Out of Range	• The value of an input variable is outside of the valid range. • The value of an input variable is incorrect.
16#2400	No Execution Right	The instruction was executed when changing the status was not possible. • While changing the settings was already in progress • While restarting the built-in EtherNet/IP port was in progress • While downloading tag data link settings from the Network Configurator is in progress
16#2402	Too Many Simultaneous Instruction Executions	Too many ChangeIPAdr, ChangeFTPAccount, and ChangeNTPServerAdr instructions were executed at the same time.

Version Information

A CPU Unit with unit version 1.02 or later and Sysmac Studio version 1.03 or higher are required to use this instruction.

FTPGetFileList

The FTPGetFileList instruction gets a list of the files in the FTP server.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FTPGetFileList	Get FTP Server File List	FB		FTPGetFileList_instance(Execute, ConnectSvr, SvrDirName, GetFileNum, SortOrder, ExecOption, RetryCfg, Cancel, FileList, Done, Busy, CommandCanceled, Error, ErrorID, ErrorIDEx, StoredNum);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Connect Svr	Connected FTP server settings	Input	Setting parameters for the connected FTP server	---	---	*1
SvrDir Name	FTP server directory name	Input	Name of FTP server directory for which to get the file list	256 bytes max. (255 single-byte alphanumeric characters plus the final NULL character)*2	---	''*3
GetFile Num	Number of files to list	Input	Number of files to list	1 to 1000	---	1
SortOrder*4	Sort order	Input	Order to sort the file list	_NAME_ASC, _NAME_DESC, _DATE_ASC, _DATE_DESC	---	_NAME _ASC
Exec Option	FTP execution options	Input	Options for FTP execution	---	---	---
RetryCfg	Execution retry settings	Input	Instruction execution retry settings	---	---	---
Cancel	Cancel	Input	TRUE: Instruction execution is canceled. FALSE: Instruction execution is not canceled.	Depends on data type.	---	FALSE

Name	Meaning	I/O	Description	Valid range	Unit	Default
FileList[] array*5*6*7	File details	In-out	Details for the obtained file list	---	---	*1
Command-Canceled	Cancel completed	Output	TRUE: Canceling completed. FALSE: Canceling not completed.	Depends on data type.	---	---
StoredNum	Number of files obtained in list		Number of files for which details were obtained	0 to 1000		

*1 If you omit an input parameter, the default value is not applied. A building error will occur.

*2 You cannot use the following characters in FTP server directory names: * ? < > | "

*3 The default is the home directory when you log onto the FTP server.

*4 If the FTP server does not support sorting names, the names are in ascending order regardless of the value of *SortOrder*.

*5 The array can have a maximum of 1,000 elements.

*6 This is a one-dimensional array. If an array with more than one dimension is specified, a building error will occur.

*7 The first array element number is 0. If a number other than 0 is specified for the first array element, a building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
ConnectSvr	Refer to *Function* for details on the structure _sFTP_CONNECT_SVR.																			
SvrDirName																				OK
GetFileNum							OK													
SortOrder	Refer to *Function* for the enumerators of the enumerated type _eFILE_SORT_ORDER.																			
ExecOption	Refer to *Function* for details on the structure _sFTP_EXEC_OPTION.																			
RetryCfg	Refer to *Function* for details on the structure _sFTP_RETRY_CFG.																			
Cancel	OK																			
FileList[] array	Refer to *Function* for details on the structure _sFTP_FILE_DETAIL.																			
Command Canceled	OK																			
StoredNum							OK													

Function

The FTPGetFileList instruction gets a list of files and file details from the specified directory *SvrDirName* on the connected FTP server *ConnectSvr*.

Specify the number of files to list in *GetFileNum*. Specify the order in which to sort the obtained file information in *SortOrder*.

The data type of *ConnectSvr* is structure _sFTP_CONNECT_SVR. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
ConnectSvr	Connected FTP server settings	Setting parameters for the connected FTP server	_sFTP_CONNECT_SVR	---	---	---
Adr	Address	IP address or host name*1	STRING	1 to 200 bytes*2	---	---
PortNo	Port number	TCP port number of FTP server control connection	UINT	0 to 65535*3		
UserName	User name	User name on FTP server	STRING	33 bytes max.*4*5*6		
Password	Password	FTP server password	STRING	33 bytes max.*4*5*6		

*1 A separate DNS or Hosts setting is required to specify a host name.

*2 You can use the following single-byte characters: A to Z, a to z, 0 to 9, - (hyphen), . (period), and _ (underbar).

*3 If you specify 0, TCP port number 21 is used.

*4 You can use the following single-byte characters: A to Z, a to z, 0 to 9, - (hyphen), . (period), and _ (underbar). You can also use \ and @ for a CPU Unit with unit version 1.16 or later.

*5 The NULL character at the end must be counted in the number of bytes.

*6 For CPU Units with unit version 1.08, specify a text string of one character or more. An error will occur if you specify a text string that contains only the final NULL character.

The data type of *SortOrder* is enumerated type _eFILE_SORT_ORDER. The meanings of the enumerators are as follows:

Enumerator	Meaning
_NAME_ASC	Ascending order of names
_NAME_DESC	Descending order of names
_DATE_ASC	Ascending order of last modified dates
_DATE_DESC	Descending order of last modified dates

The file details is stored in *FileList[]*. The number of files for which information was obtained is stored in *StoredNum*.

The data type of *FileList[]* is structure _sFTP_FILE_DETAIL. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
FileList	File details	Details for the obtained file list	_sFTP_FILE _DETAIL	---	---	---
Name	File or folder name	File or folder name	STRING	256 bytes max. (255 single-byte alphanumeric characters plus the final NULL character)	---	---
ModifiedDate	Last modified date	The last modified data of the file or folder	DATE_AND_ TIME	---		
Size	File size	The file size*1	ULINT		Bytes	
ReadOnly	Read-only attribute	The read-only attribute of the file or folder TRUE: Read only FALSE: Not read only	BOOL	Depends on data type.	---	
Folder	Folder	TRUE: Folder FALSE: Not a folder	BOOL			

*1 The file size is 0 for a folder.

Specifying Options for FTP Server Processing

The operation specified with *ExecOption* is performed to obtain the file list from the FTP server.

The option settings are the same as those for the FTPGetFile instruction (page 2-1128). Refer to the specified page for details.

However, the only option that is valid for this instruction is *ExecOption.PassiveMode*.

Specifying Retrying Connection Processing with the FTP Server

Connection processing with the FTP server times out and ends when the specified timeout time *RetryCfg.TimeOut* is exceeded before a connection is successfully established. If the FTP server rejects the connection, processing ends before reaching the timeout time. After failing to connect, connection processing is retried after the specified retry interval *RetryInterval*. If a connection with the FTP server is not established within the number of retries specified with *RetryCfg.RetryNum*, an instruction execution error occurs.

If, after a successful connection to the FTP server, a problem occurs on the network that interrupts file transfer for longer than the time specified with *RetryCfg.TimeOut*, a timeout occurs and retry processing is not performed.

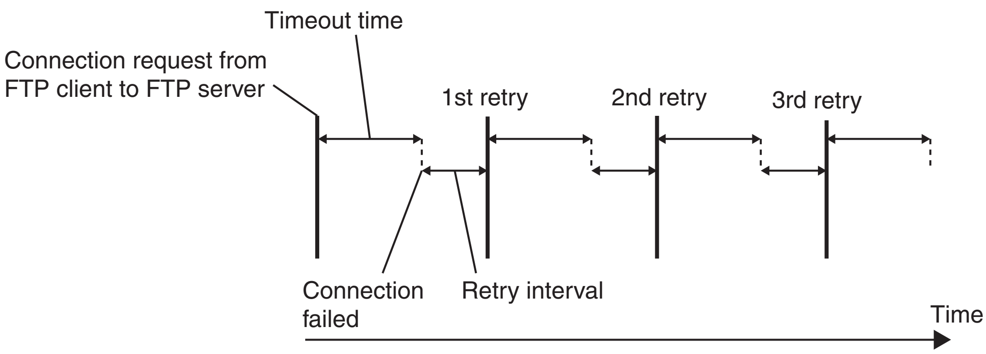
The data type of *RetryCfg* is structure _sFTP_RETRY_CFG. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
RetryCfg	Execution retry settings	Instruction execution retry settings	_sFTP_RETRY_CFG	---	---	---
TimeOut	Timeout time	Timeout time for a connection to the FTP server	UINT	0 to 60*1	Seconds	20
RetryNum	Number of retries	The number of retries when connection fails	UINT	0 to 3	Times	0
RetryInterval	Retry interval	The interval for retrying when connection fails	UINT	0 to 65535*2	Seconds	1

*1 If 0 is set, the timeout time is 20 s.

*2 If 0 is set, the retry interval is 1 s.

The following figure shows the relation between the timeout time, number of retries, and retry interval when an FTP client performs connection processing to a FTP server.

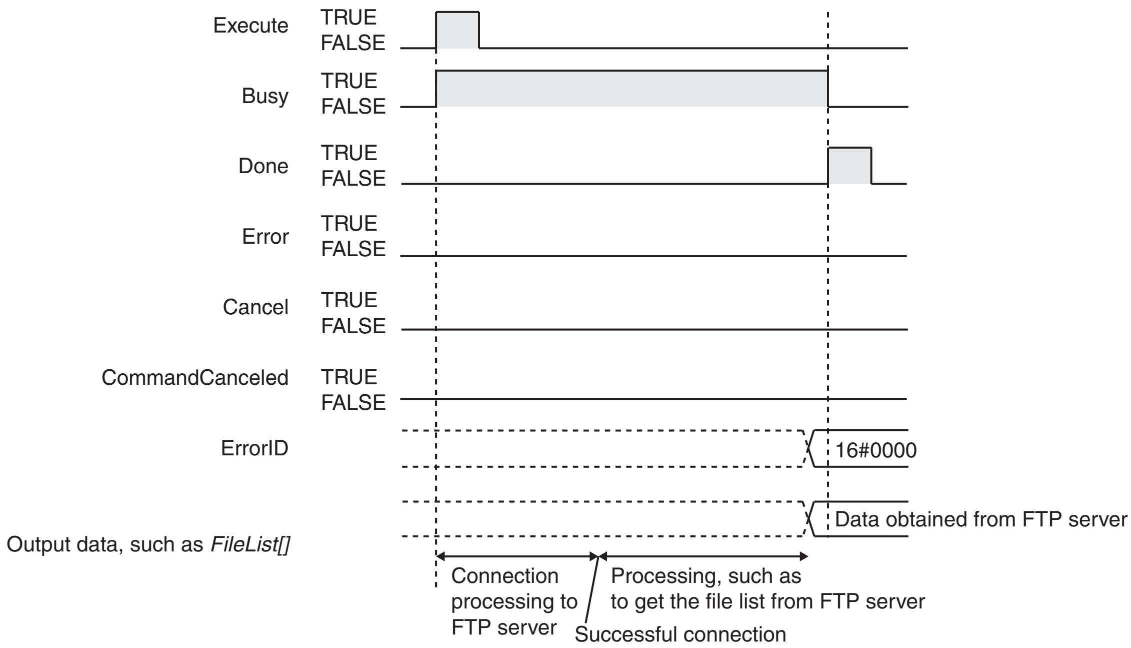


Successfully Connecting to the FTP Server

When connection processing to the FTP server is successfully completed and the file list is obtained from the FTP server, the following processing is performed.

- A value of 16#0000 is stored in *ErrorID*.
- The obtained data is stored in the output data, such as *FileList[]*.
- The value of *Done* is changed to TRUE.

The following timing chart is an example for successful connection processing to the FTP server.

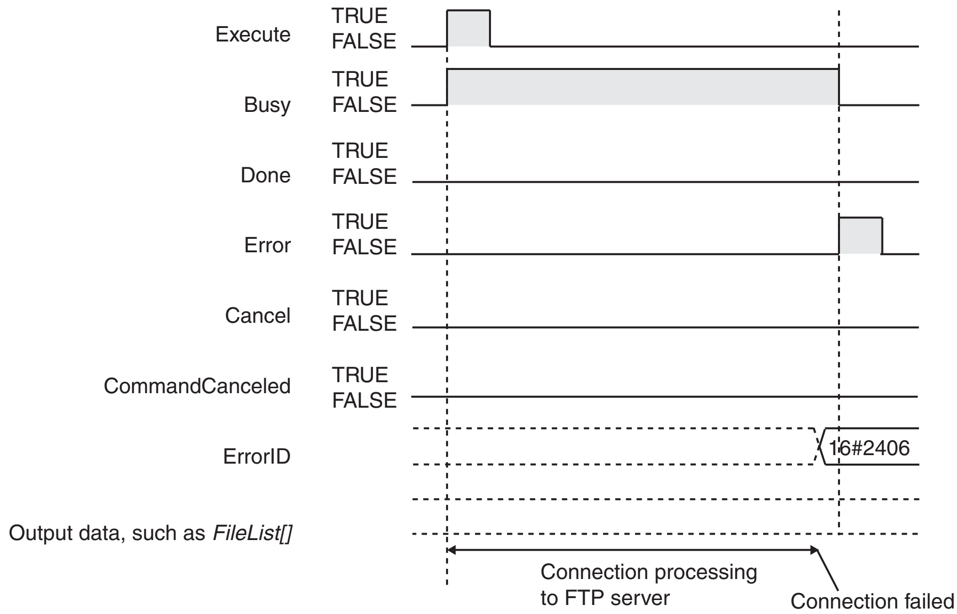


● Failing to Connect to the FTP Server

The following processing is performed when connection processing to the FTP server fails.

- The error code is stored in *ErrorID*.
- The value of *Error* is changed to TRUE.

The following timing chart is an example for when connection processing to the FTP server fails.



Canceling Instruction Execution

If *Cancel* changes to TRUE during instruction execution, processing with the FTP server is forced to end. You can use it to end processing when obtaining the file list or connection processing to the FTP server is taking too much time.

● When *Cancel* Changes to TRUE during Processing with the FTP Server

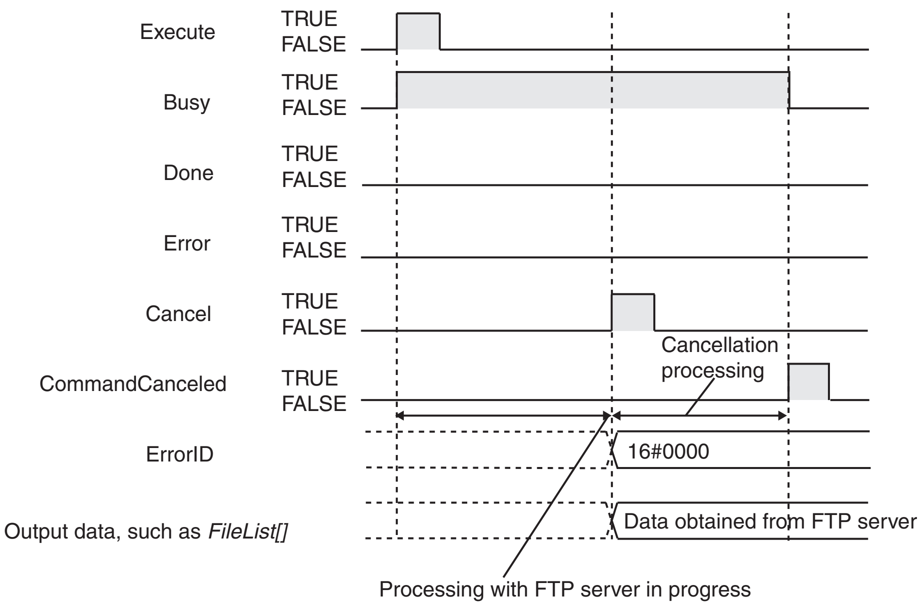
The following occurs if *Cancel* changes to TRUE while the FTPGetFileList instruction is obtaining a file list from the FTP server.

Any file details that were obtained from the FTP server is stored in *FileList[]*.

The number of files for which file details were correctly obtained is stored in *StoredNum*.

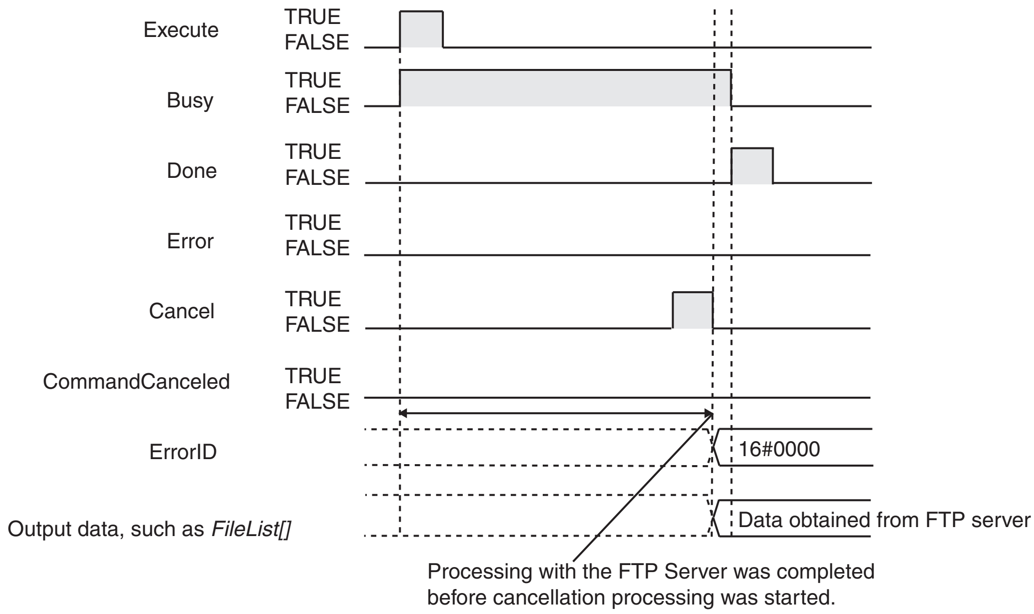
The value of *Done* does not change to TRUE.

The value of *CommandCanceled* changes to TRUE when cancellation is completed. Use this to confirm normal completion of cancellation.



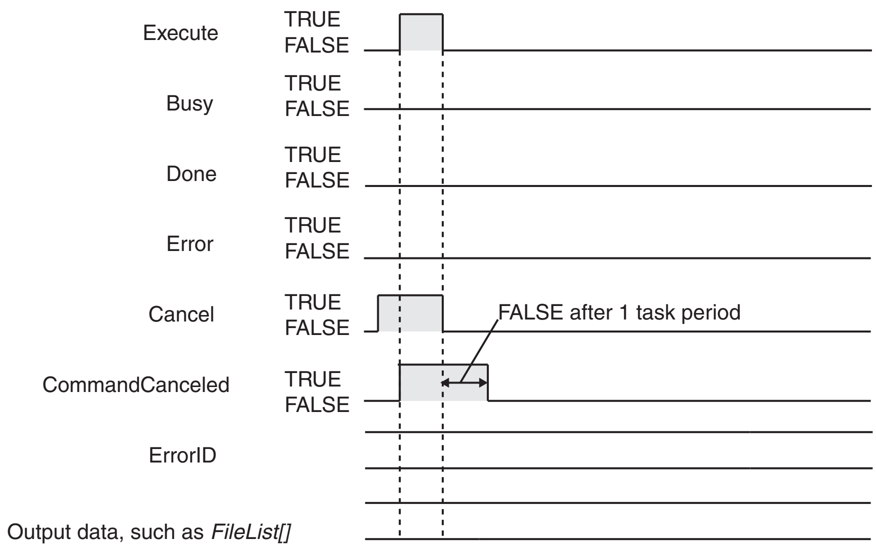
When Processing with the FTP Server Is Completed Before Cancellation Processing Is Started

Even if *Cancel* is changed to TRUE, *Done* changes to TRUE to indicate normal completion if processing with the FTP server is completed before cancellation processing is started. The value of *CommandCanceled* does not change to TRUE.



When both *Cancel* and *Execute* Are TRUE

If both *Cancel* and *Execute* are TRUE, cancellation is given priority and processing is not performed with the FTP server. *CommandCanceled* changes to TRUE.



Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta*1	Online	BOOL	This variable indicates when built-in EtherNet/IP port communications can be used. TRUE: Communications are possible. FALSE: Communications are not possible.
_EIP1_EtnOnlineSta*2 _EIP2_EtnOnlineSta*3			
_EIPIn1_EtnOnlineSta*4			

*1 Use this variable name for an NJ-series CPU Unit.

*2 Use this variable name for port 1 on an NX-series CPU Unit, or for an NY-series Controller.

*3 Use this variable name for port 2 on an NX-series CPU Unit.

*4 Use this variable name for the internal communication port on an NY-series Controller.

Precautions for Correct Use

- This instruction can be used only for the built-in EtherNet/IP ports on NJ/NX-series CPU Units and NY-series Controllers.
- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* (page 2-3) for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- If there are no files or subdirectories in the directory specified by the *SvrDirName* input variable, *Done* changes to TRUE to indicate a normal end. If 0 is stored in *StoredNum*, nothing is stored in *FileList[]*.
- If the number of array elements in *FileList[]* is less than the number of files specified with the *GetFileNum* input variable, only the file information that will fit in *FileList[]* is stored and the file information that does not fit is not stored. In this case, *Error* does not change to TRUE.
- If a file name exceeds 255 characters, the first 255 characters are stored in *Name* in *FileList[]*. In this case, *Error* does not change to TRUE.
- It may be impossible to obtain some or all of the specified file details depending on FTP server specifications. The members of *FileList[]* take the following values for files for which details are not obtained. In this case, the value of *Error* is FALSE.

Member	Value
ModifiedDate	DT#1970-01-01-00:00:00.000000000
Size	0
ReadOnly	FALSE
Folder	FALSE

- You can execute a maximum of 3 of the following instructions at the same time: FTPGetFileList, FTPGetFile, FTPPutFile, FTPRemoveFile, and FTPRemoveDir.
- An error will occur in the following cases. *Error* will change to TRUE.
  - The value of any input parameter is outside of the valid range.
  - “..” is specified for a directory level in *SvrDirName*.
  - An incorrect path such as “//” is specified for *SvrDirName*.
  - The directory specified by *SvrDirName* does not exist on the FTP server.
  - The FTP server specified by *ConnectSvr* does not exist on the network or the specified FTP server is not operating.
  - More than 3 of the following instructions were executed at the same time: FTPGetFileList, FTPGetFile, FTPPutFile, FTPRemoveFile, and FTPRemoveDir.

- File transfer processing was interrupted during FTP server connection processing by a problem on the network.

- For this instruction, expansion error code *ErrorIDEx* gives the FTP response code that was returned by the FTP server. The following table lists typical values of *ErrorIDEx* and describes the meanings of the errors and the corrections. For details, refer to FTP server specifications. An expansion error code is output to *ErrorIDEx* when the value of error code *ErrorID* is WORD#16#2407.

Value of *ErrorI-DEx*	Meaning	Correction
16#000001A9	It was not possible to establish a data connection.	If you use FTP communications with an FTP server over the Internet, make sure that the FTP open mode is not set to active.
16#000001AA	The connection was closed. Data transfer was aborted.	Check the connection to the FTP server. Make sure that the FTP server is operating.
16#000001C2	It was not possible to perform the requested file operation. Using the file was not possible, e.g., it is already open.	Make sure that the file is not open for any other application.
16#00000212	User login was not possible.	Check the FTP user name and password.
16#00000214	An account to save files is required.	Check the FTP user access rights.
16#00000226	Execution of the requested file operation was not possible because using the file was not possible, e.g., accessing it was not possible because it was not found.	Make sure that a file with the specified name exists in the directory on the FTP server. Check the access rights of the specified file.
16#00000229	Execution was not possible because the file name was not correct.	Check the access rights of the specified directory.

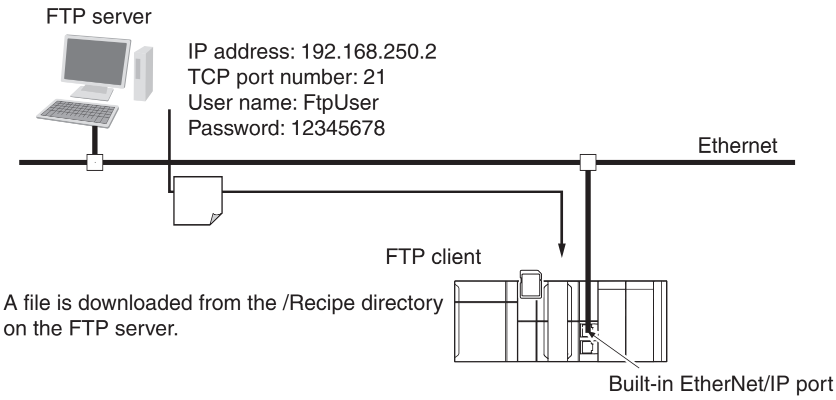
Version Information

A CPU Unit with unit version 1.08 or later and Sysmac Studio version 1.09 or higher are required to use this instruction.

Sample Programming

The following programming downloads a file from the /Recipe directory on the FTP server and stores it in the root directory of an SD Memory Card.

The file to download is the last file in the /Recipe directory on the FTP server when the files are sorted in ascending order of names.



The Controller is connected to the FTP server through an EtherNet/IP network. The settings of the parameters to connect to the FTP server are given in the following table.

Parameter	Value
IP address	192.168.250.2
TCP port number	21
User name	FtpUser
Password	12345678

The following procedure is used.

1 The FTPGetFileList instruction is used to get a file list from the FTP server. The following table gives the FTP server directory name, number of files to list, sort order, and variable to store file details.

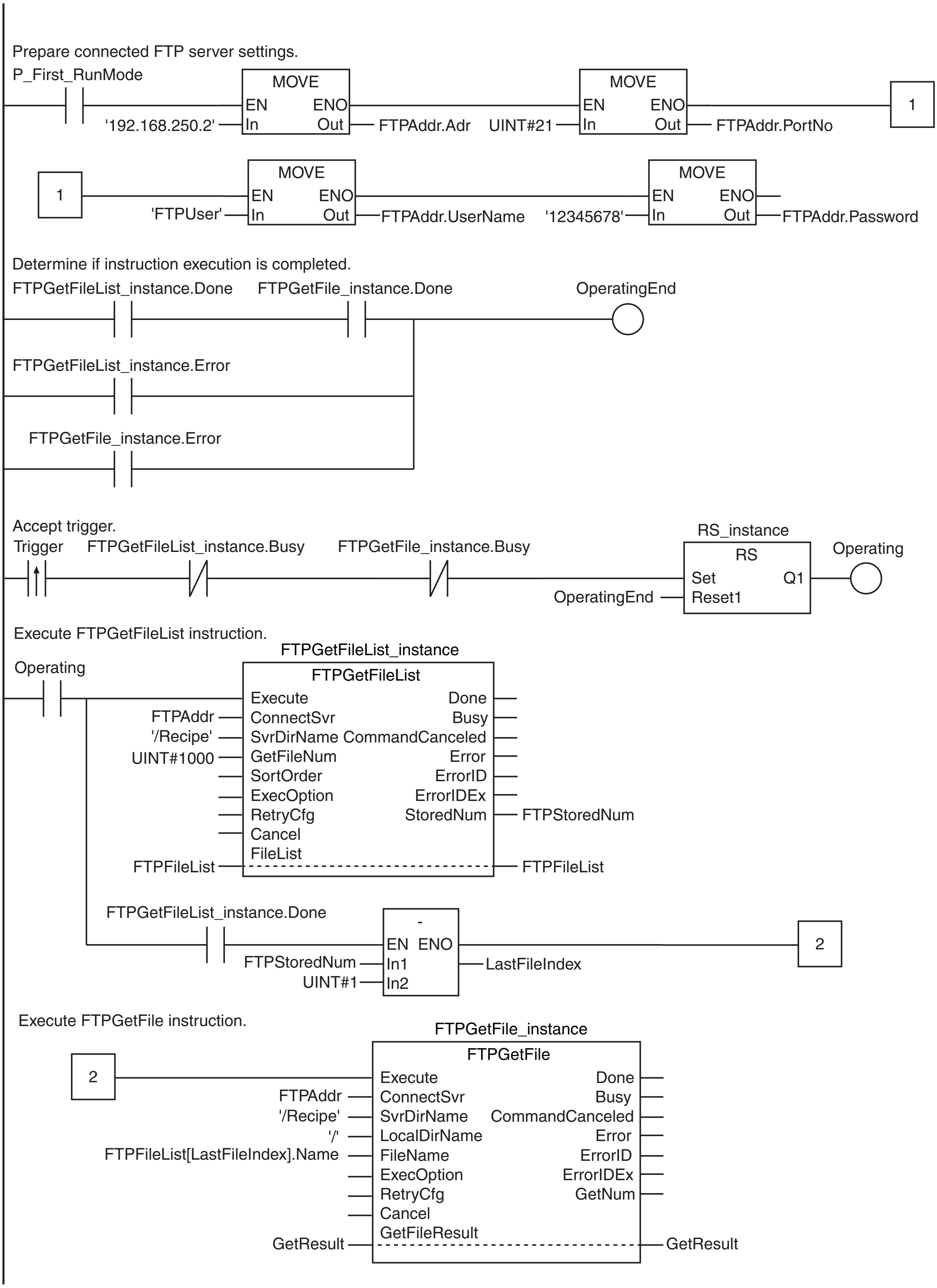
Item	Specification
FTP server directory name	'/Recipe'
Number of files to list	1000
Sort order	Ascending order of names
Variable to store file details	FTPFileList[]

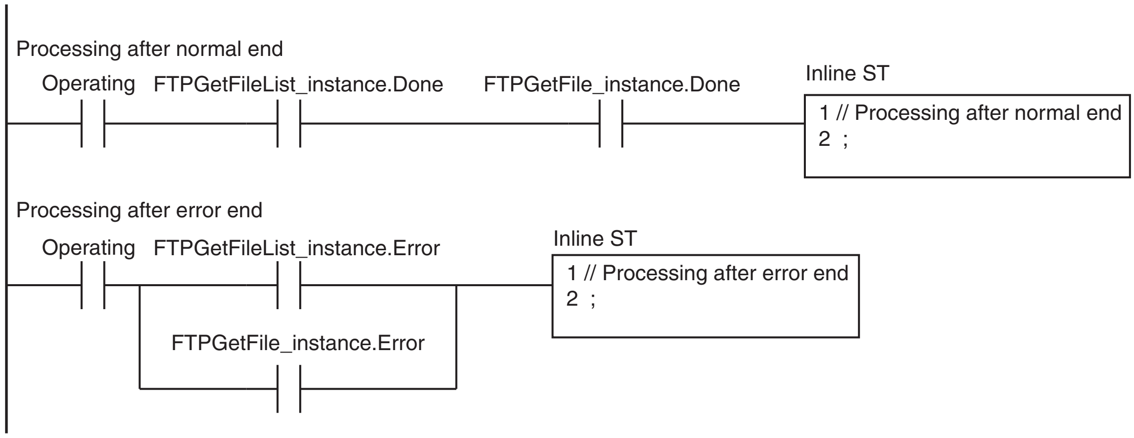
2 The FTPGetFile instruction is used to download the last file from the file list obtained in step 1 when the list is in ascending order of names. The file is stored in the root directory on the SD Memory Card.

3 Normal end processing is executed if all processing ends normally. Processing for an error end is performed if an error occurs.

LD

Internal Vari-ables	Variable	Data type	Initial value	Comment
	FTPGetFileL-ist_instance	FTPGetFileList		Instance of FTPGetFileL-ist instruction
	FTPGetFile_in-stance	FTPGetFile		Instance of FTPGetFile instruction
	FTPAddr	_sFTP_CON-NECT_SVR	(Adr := '', PortNo := 0, User-Name := '', Password := '')	Connected FTP server settings
	FTPFileList	ARRAY[0..999] OF _sFTP_FILE_DE-TAIL	[1000((Name := '', Modified-Date := DT#1970-01-01-00:00:00, Size := 0, ReadOnly := False, Folder := False))]	File details
	GetResult	ARRAY[0..0] OF _sFTP_FILE_RE-SULT	[(Name := '', TxError := False, RemoveError := False, Reserved := [4(16#0)])]	Downloaded file results
	FTPStoredNum	UINT	0	Number of files obtained in file list
	LastFileIndex	UINT	0	Index of last file when list is in ascending order of names
	RS_instance	RS		Instance of RS instruction
	OperatingEnd	BOOL	FALSE	Processing completed
	Trigger	BOOL	FALSE	Execution condition
	Operating	BOOL	FALSE	Processing


Prepare connected FTP server settings.
P_First_RunMode
MOVE
EN ENO
In Out
'192.168.250.2'
FTPAddr.Adr
UINT#21
MOVE
EN ENO
In Out
FTPAddr.PortNo
1
1
MOVE
EN ENO
In Out
'FTPUser'
FTPAddr.UserName
'12345678'
MOVE
EN ENO
In Out
FTPAddr.Password
Determine if instruction execution is completed.
FTPGetFileList_instance.Done
FTPGetFile_instance.Done
OperatingEnd
FTPGetFileList_instance.Error
FTPGetFile_instance.Error
Accept trigger.
Trigger
FTPGetFileList_instance.Busy
FTPGetFile_instance.Busy
RS_instance
RS
Set Q1
Reset1
OperatingEnd
Operating
Execute FTPGetFileList instruction.
FTPGetFileList_instance
Operating
FTPGetFileList
Execute Done
ConnectSvr Busy
SvrDirName CommandCanceled
GetFileNum Error
SortOrder ErrorID
ExecOption ErrorIDEx
RetryCfg StoredNum
Cancel
FileList
FTPAddr
'/Recipe'
UINT#1000
FTPFileList
FTPStoredNum
FTPFileList
FTPGetFileList_instance.Done
-
EN ENO
In1
In2
FTPStoredNum
UINT#1
LastFileIndex
2
Execute FTPGetFile instruction.
FTPGetFile_instance
2
FTPGetFile
Execute Done
ConnectSvr Busy
SvrDirName CommandCanceled
LocalDirName Error
FileName ErrorID
ExecOption ErrorIDEx
RetryCfg GetNum
Cancel
GetFileResult
FTPAddr
'/Recipe'
'/'
FTPFileList[LastFileIndex].Name
GetResult
GetResult



Processing after normal end
Operating
FTPGetFileList_instance.Done
FTPGetFile_instance.Done
Inline ST
1 // Processing after normal end
2 ;
Processing after error end
Operating
FTPGetFileList_instance.Error
FTPGetFile_instance.Error
Inline ST
1 // Processing after error end
2 ;

ST

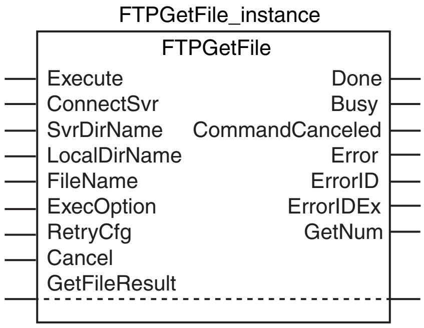
Internal Vari-ables	Variable	Data type	Initial value	Comment
	R_TRIG_instance	R_TRIG		Instance of R_TRIG instruction
	UP_Q	BOOL	FALSE	Trigger output
	FTPGetFile_in-stance	FTPGetFile		Instance of FTPGetFile instruction
	FTPGetFileL-ist_instance	FTPGetFileList		Instance of FTPGetFileL-ist instruction
	FTPFileList	ARRAY[0..999] OF _sFTP_FILE_DE-TAIL	[1000((Name := '', Modified-Date := DT#1970-01-01-00:00:00, Size := 0, ReadOnly := False, Folder := False))]	File details
	FTPStoredNum	UINT	0	Number of files obtained in file list
	DoFTPTrigger	BOOL	FALSE	Execution condition for FTPGetFileList and FTP-GetFile
	FTPAddr	_sFTP_CON-NECT_SVR	(Adr := '', PortNo := 0, User-Name := '', Password := '')	Connected FTP server settings
	GetResult	ARRAY[0..0] OF _sFTP_FILE_RE-SULT	[(Name := '', TxError := False, RemoveError := False, Reserved := [4(16#0)])]	Downloaded file results
	Stage	UINT	0	Instruction execution stage
	Trigger	BOOL	FALSE	Execution condition

```
// Prepare connected FTP server settings.
IF P_First_RunMode THEN
   FTPAddr.Adr      := '192.168.250.2'; // Address
   FTPAddr.PortNo   := UINT#21;         // Port number
   FTPAddr.UserName := 'FtpUser';       // User name
   FTPAddr.Password := '12345678';      // Password
END_IF;

Accept trigger.
R_TRIG_instance(Trigger, UP_Q);
IF ( (UP_Q = TRUE) AND (FTPGetFileList_instance.Busy = FALSE) AND
     (FTPGetFile_instance.Busy = FALSE) ) THEN
   DoFTPTrigger := TRUE;
   Stage := INT#1;
   FTPGetFileList_instance( // Initialize instance.
      Execute    := FALSE,
      ConnectSvr := FTPAddr,
      SvrDirName := '/Recipe',
      GetFileNum := UINT#1000,
      FileList   := FTPFileList,
      StoredNum  => FTPStoredNum) ;
   FTPGetFile_instance(  // Initialize instance.
      Execute       := FALSE,
      ConnectSvr    := FTPAddr,
      SvrDirName    := '/Recipe',
      LocalDirName  := '/',
      FileName      := '',
      GetFileResult := GetResult) ;
END_IF;
```

```
IF (DoFTPTrigger = TRUE) THEN
   CASE Stage OF
      1 : // Execute FTPGetFileList instruction
         FTPGetFileList_instance(
            Execute    := TRUE,           // Execution
            ConnectSvr := FTPAddr,        // Connected FTP server
            SvrDirName := '/Recipe',      // FTP server directory name
            GetFileNum := UINT#1000,      // Number of files to list
            FileList   := FTPFileList,    // File details
            StoredNum  => FTPStoredNum) ;// Number of files obtained in list
         IF (FTPGetFileList_instance.Done = TRUE) THEN
            Stage := INT#2;               // To next stage
         ELSIF (FTPGetFileList_instance.Error = TRUE) THEN
            Stage := INT#10;              // Error end
         END_IF;
      2 : // Execute FTPGetFile instruction.
         FTPGetFile_instance(
            Execute       := TRUE,
            // Execution
            ConnectSvr    := FTPAddr,
            // Connected FTP server
            SvrDirName    := '/Recipe',
            // FTP server directory name
            LocalDirName  := '/',
            // Local directory name
            FileName      := FTPFileList[FTPStoredNum - 1].Name,
            // File name
            GetFileResult := GetResult) ;
            // Downloaded file results
         IF (FTPGetFile_instance.Done = TRUE) THEN
            Stage := INT#0;                                    // Normal end
         ELSIF (FTPGetFile_instance.Error = TRUE) THEN
            Stage := INT#20;                                  // Error end
         END_IF;
      0:  // Processing after normal end
         DoFTPTrigger:=FALSE;
         Trigger     :=FALSE;
      ELSE // Processing after error end
         DoFTPTrigger:=FALSE;
         Trigger     :=FALSE;
   END_CASE;
END_IF;
```

FTPGetFile

The FTPGetFile instruction downloads a file from the FTP server.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FTPGetFile	Get File from FTP Server	FB		FTPGetFile_instance(Execute, ConnectSvr, SvrDirName, LocalDirName, FileName, ExecOption, RetryCfg, Cancel, GetFileResult, Done, Busy, CommandCanceled, Error, ErrorID, ErrorIDEx, GetNum) ;

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Connect Svr	Connected FTP server settings	Input	Setting parameters for the connected FTP server	---	---	*1
SvrDir Name	FTP server directory name		Name of FTP server directory from which to download a file	256 bytes max. (255 single-byte alphanumeric characters plus the final NULL character)*2		''*3
LocalDir Name	Local directory name		Name of the directory in which to store the file downloaded from the FTP server	256 bytes max. (255 single-byte alphanumeric characters plus the final NULL character)		'/'
FileName	File name		Name of file to download*4	256 bytes max. (255 single-byte alphanumeric characters plus the final NULL character)*5		*1
Exec Option	FTP execution options		Options for FTP execution	---		---
RetryCfg	Execution retry settings		Instruction execution retry settings			
Cancel	Cancel		TRUE: Instruction execution is canceled. FALSE: Instruction execution is not canceled.	Depends on data type.		FALSE

Name	Meaning	I/O	Description	Valid range	Unit	Default
GetFile Result[] array*6*7*8	Downloaded file results	In-out	Downloaded file results	---	---	*1
Command-Canceled	Cancel com-pleted	Output	TRUE: Canceling completed. FALSE: Canceling not com-pleted.	Depends on data type.	---	---
GetNum	Number of files to down-load		Number of files to download	---		

*1 If you omit an input parameter, the default value is not applied. A building error will occur.

*2 You cannot use the following characters in FTP server directory names: * ? < > | "

*3 The default is the home directory when you log onto the FTP server.

*4 You can use wildcards in file names.

*5 You cannot use the following character in file names: |

*6 The array can have a maximum of 1,000 elements.

*7 This is a one-dimensional array. If an array with more than one dimension is specified, a building error will occur.

*8 The first array element number is 0. If a number other than 0 is specified for the first array element, a building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
ConnectSvr	Refer to *Function* for details on the structure _sFTP_CONNECT_SVR.																			
SvrDirName																				OK
LocalDirName																				OK
FileName																				OK
ExecOption	Refer to *Function* for details on the structure _sFTP_EXEC_OPTION.																			
RetryCfg	Refer to *Function* for details on the structure _sFTP_RETRY_CFG.																			
Cancel	OK																			
GetFileResult[] array	Refer to *Function* for details on the structure _sFTP_FILE_RESULT.																			
Command Canceled	OK																			
GetNum							OK													

Function

The FTPGetFile instruction downloads the file specified with *FileName* from the specified directory *SvrDirName* on the connected FTP server *ConnectSvr* to the directory specified with *LocalDirName* in the SD Memory Card. If the specified directory *LocalDirName* does not exist in the SD Memory Card, a new directory is created and the specified file is downloaded.

You can use wildcards in *FileName*. This allows you to download more than one file at one time.

The results of downloading is stored in *GetFileResult[]* for each file. Store the number of files to download in *GetNum*. If you use a wildcard in *FileName*, store the number of files with names that match the wildcard.

If the actual number of transferred files is different, the value of *GetFileResult[].TxError* changes to TRUE.

If an error occurs when deleting the source file after the download, the value of *GetFileResult[].RemoveError* changes to TRUE.

With an NY-series Controller, files are downloaded into the shared folder (Virtual SD Memory Card). Before downloading files to the Virtual SD Memory Card, you must make the settings for the Virtual SD Memory Card. Refer to the *NY-series Industrial Panel PC / Industrial Box PC Software User's Manual* (Cat. No. W558) for details on a Virtual SD Memory Card.

The data type of *ConnectSvr* is structure _sFTP_CONNECT_SVR. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
ConnectSvr	Connected FTP server settings	Setting parameters for the connected FTP server	_sFTP_CONNECT_SVR	---	---	---
Adr	Address	IP address or host name*1	STRING	1 to 200 bytes*2	---	---
PortNo	Port number	TCP port number of FTP server control connection	UINT	0 to 65535*3		
UserName	User name	User name on FTP server	STRING	33 bytes max.*4*5*6		
Password	Password	FTP server password	STRING	33 bytes max.*4*5*6		

*1 A separate DNS or Hosts setting is required to specify a host name.

*2 You can use the following single-byte characters: A to Z, a to z, 0 to 9, - (hyphen), . (period), and _ (underbar).

*3 If you specify 0, TCP port number 21 is used.

*4 You can use the following single-byte characters: A to Z, a to z, 0 to 9, - (hyphen), . (period), and _ (underbar). You can also use \ and @ for a CPU Unit with unit version 1.16 or later.

*5 The NULL character at the end must be counted in the number of bytes.

*6 For CPU Units with unit version 1.08, specify a text string of one character or more. An error will occur if you specify a text string that contains only the final NULL character.

The data type of *GetFileResult[]* is structure _sFTP_FILE_RESULT. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
GetFileResult	Downloaded file results	Transferred file results	_sFTP_FILE_RESULT	---	---	---
Name	File name*1	Transferred file name	STRING	256 bytes max. (255 single-byte alphanumeric characters plus the final NULL character)	---	---
TxError	Transfer error	TRUE: Transfer ended in an error. FALSE: Transfer ended normally.	BOOL	Depends on data type.		
RemoveError	Deletion error	TRUE: Deletion ended in an error. FALSE: Deletion ended normally.	BOOL			
Reserved	Reserved	Reserved by the system.	ARRAY[0..3] Of Byte	---		0

*1 The file name extension is included.

Using Wildcards to Specify File Names

You can use wildcards to specify the names of the files to download in *FileName*.

As wildcards, you can specify “*” and “?”. “*” represents one or more characters. “?” represents only one character.

Examples of using wildcard specifications are given below.

Assume that the FTP server directory has the following file structure.

```
└─DataFiles (specified directory)
    LogA01.log
    LogA02.txt
    LogB.log
    LogC.txt
    ControlDataA1.csv
    ControlDataA10.csv
    ControlDataA100.csv
    ControlDataB10.csv
    ControlDataC100.csv

  └─ControlSubDataFiles (subdirectory)
      SubData_A001.txt
      SubData_A002.txt
```

As shown in the following table, the way that the wildcards are used determines the files that are specified.

Wildcard specification	Specified files
Log*.log	LogA01.log, LogB.log
Log?.log	LogB.log
Log?.*	LogB.log, LogC.txt
Data	ControlDataA1.csv, ControlDataA10.csv, ControlDataA100.csv, ControlDataB10.csv, ControlDataC100.csv, (ControlSubDataFiles)*1
*	All files except for those in the subdirectory
.	All files except for those in the subdirectory
?.?	No files
????.???	LogB.log, LogC.txt

*1 Subdirectory files will also be included for some FTP server specifications.

If you specify wildcards, you can download up to 1,000 files.

If *GetFileResult[].TxError* or *GetFileResult[].RemoveError* is TRUE as the result of downloading files, *Error* changes to TRUE, the corresponding error code for the first error is stored in *ErrorID* and the error response from the FTP server is stored in *ErrorIDEx*.

Specifying Options for FTP Server Processing

The operation specified with *ExecOption* is performed to download files from the FTP server.

The data type of *ExecOption* is structure _sFTP_EXEC_OPTION. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
ExecOption	FTP execution options	Options for FTP execution	_sFTP_EXEC_OPTION	---	---	---
PassiveMode	Passive mode specification	TRUE: Passive mode FALSE: Active mode	BOOL	Depends on data type.	---	FALSE
ASCIIMode	ASCII mode specification	TRUE: ASCII mode FALSE: Binary mode	BOOL			
FileRemove	File deletion after transfer specification*1	TRUE: Delete files after transfer. FALSE: Do not delete files after transfer.	BOOL			
OverWrite	Overwrite specification	TRUE: Overwrite files at transfer destination. FALSE: Do not overwrite files at transfer destination.	BOOL			
Reserved	Reserved	Reserved by the system.	ARRAY[0..7] Of Byte	---		0

*1 The transfer source files are not deleted when the transfer fails.

The meanings of the options are described next.

● *PassiveMode* (Passive Mode Specification)

The passive mode specification tells whether to use passive mode for the data connection request to the FTP server. If passive mode is not specified, active mode is used for the data connection request to the FTP server.

Refer to the *NJ/NX-series CPU Unit Built-in EtherNet/IP Port User's Manual* (Cat. No. W506) or *NY-series Industrial Panel PC / Industrial Box PC Built-in EtherNet/IP Port User's Manual* (Cat. No. W563) for details on connection request methods.

The values and their meanings for *PassiveMode* are given in the following table.

Set value	Meaning
TRUE	The data connection request with the FTP server is performed in passive mode. The data connection request is performed from the FTP client.
FALSE	The data connection request with the FTP server is performed in active mode. The data connection request is performed from the FTP server.

● *ASCIIMode* (ASCII Mode Specification)

The ASCII mode specification tells whether ASCII mode is used as the transfer mode from the transfer source system to the transfer destination system. If ASCII mode is not specified, binary mode is used as the transfer mode from the transfer source system to the transfer destination system.

The values and their meanings for *ASCIIMode* are given in the following table.

Set value	Meaning
TRUE	ASCII mode is used as the transfer mode from the transfer source system to the transfer destination system. Text line feed codes are converted from those for the transfer source system to those for the transfer destination system.
FALSE	Binary mode is used as the transfer mode from the transfer source system to the transfer destination system. Text line feed codes are transferred as is from the transfer source system.

● *FileRemove* (File Deletion after Transfer Specification)

The file deletion after transfer specification tells whether to delete the transfer source files after they are transferred to the transfer destination.

The values and their meanings for *FileRemove* are given in the following table.

Set value	Meaning
TRUE	The transfer source files are deleted.
FALSE	The transfer source files are not deleted.

● *OverWrite* (Overwrite Specification)

The overwrite specification tells whether to overwrite files with the same name at the transfer destination when files are downloaded. If overwriting is not specified, files with the same name as those at the transfer destination are not transferred.

File names are not case sensitive.

The values and their meanings for *OverWrite* are given in the following table.

Set value	Meaning
TRUE	The transfer destination files are overwritten.
FALSE	The transfer destination files are not overwritten. The files are not transferred to the transfer destination.

Specifying Retrying Connection Processing with the FTP Server

You can specify retrying connection processing with the FTP server.

The operation for the retry settings is the same as that for the FTPGetFileList instruction (page 2-1111). Refer to the specified page for details.

Canceling Instruction Execution

You can cancel execution of the FTPGetFile instruction after execution has started.

The results of downloading files from the FTP server up to the point where it is canceled are stored in *GetNum* and *GetFileResult[]*.

The operation for cancellation is the same as that for the FTPGetFileList instruction (page 2-1111). Refer to the specified page for details.

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta*1	Online	BOOL	This variable indicates when built-in EtherNet/IP port communications can be used. TRUE: Communications are possible. FALSE: Communications are not possible.
_EIP1_EtnOnlineSta*2 _EIP2_EtnOnlineSta*3			
_EIPIn1_EtnOnlineSta*4			
_Card1Ready	SD Memory Card Ready Flag	BOOL	This variable indicates whether the SD Memory Card is recognized and usable. TRUE: Can be used. FALSE: Cannot be used.

*1 Use this variable name for an NJ-series CPU Unit.

*2 Use this variable name for port 1 on an NX-series CPU Unit, or for an NY-series Controller.

*3 Use this variable name for port 2 on an NX-series CPU Unit.

*4 Use this variable name for the internal communication port on an NY-series Controller.

Precautions for Correct Use

- This instruction can be used only for the built-in EtherNet/IP ports on NJ/NX-series CPU Units and NY-series Controllers.
- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to Using this Section (page 2-3) for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- If the number of downloaded file results to store exceeds the number of array elements in *GetFileResult[]*, the results that will not fit are not stored. In this case, *Error* does not change to TRUE.
- If a file name exceeds 255 characters, the first 255 characters are stored in *Name* in *GetFileResult[]*. In this case, *Error* does not change to TRUE.
- You can execute a maximum of 3 of the following instructions at the same time: FTPGetFileList, FTPGetFile, FTPPutFile, FTPRemoveFile, and FTPRemoveDir.
- If a wildcard is used in the file name and an error occurs for more than one file, the results of the first file for which the value of *GetFileResult[].TxError* is TRUE of all the files for which results are stored in *GetFileResult[]* are stored in *ErrorID* and *ErrorIDEx*.
- File names are not case sensitive. Therefore, if the only difference between the names of the files at the transfer destination and the transfer files is in capitalization, the files are detected as having the same names. The following is performed in this case.

Value of *OverWrite*	Overwrite specification	Processing
TRUE	Overwrite	The files are overwritten.
FALSE	Do not overwrite.	The transfer destination files are not overwritten. The files are not transferred to the transfer destination.

- If the file specified by *FileName* does not exist in the specified directory on the FTP server, a transfer error occurs and the value of *GetFileResult[].TxError* changes to TRUE.
- If the name specified for *FileName* is actually the name of a directory, a transfer error occurs and the value of *GetFileResult[].TxError* changes to TRUE.
- If *ExecOption.FileRemove* is TRUE and the file specified with *FileName* has a read-only attribute, a deletion error occurs and *GetFileResult[].RemoveError* changes to TRUE.

- An error will occur in the following cases. *Error* will change to TRUE.
  - The value of any input parameter is outside of the valid range.
  - “..” is specified for a directory level in *SvrDirName* or *LocalDirName*.
  - An incorrect path such as “//” is specified for *SvrDirName* or *LocalDirName*.
  - The directory specified by *SvrDirName* does not exist on the FTP server.
  - More than 1,000 files to download exist in the FTP server directory specified with *SvrDirName*.
  - The file directory specified with *FileName* does not exist in the download source directory on the FTP server.
  - *ExecOption.OverWrite* is FALSE and a file with the same name as the specified file name *FileName* already exists in the specified directory *SvrDirName*.
  - *ExecOption.FileRemove* is TRUE but a file with a name that matches *FileName* has a read-only attribute.
  - The FTP server specified by *ConnectSvr* does not exist on the network or the specified FTP server is not operating.
  - Accessing the file specified with *FileName* failed because there is no access right to the file or the file is corrupted.
  - More than 3 of the following instructions were executed at the same time: FTPGetFileList, FTPGetFile, FTPPutFile, FTPRemoveFile, and FTPRemoveDir.
  - The SD Memory Card is not in a usable condition.
  - The SD Memory Card is write protected.
  - There is insufficient space available on the SD Memory Card.
  - The maximum number of files or directories was exceeded on the SD Memory Card.
- For this instruction, expansion error code *ErrorIDEx* gives the FTP response code that was returned by the FTP server. The following table lists typical values of *ErrorIDEx* and describes the meanings of the errors and the corrections. For details, refer to FTP server specifications. An expansion error code is output to *ErrorIDEx* when the value of error code *ErrorID* is WORD#16#2407.

Value of *ErrorIDEx*	Meaning	Correction
16#000001A9	It was not possible to establish a data connection.	If you use FTP communications with an FTP server over the Internet, make sure that the FTP open mode is not set to active.
16#000001AA	The connection was closed. Data transfer was aborted.	Check the connection to the FTP server. Make sure that the FTP server is operating.
16#000001C2	It was not possible to perform the requested file operation. Using the file was not possible, e.g., it is already open.	Make sure that the file is not open for any other application.
16#00000212	User login was not possible.	Check the FTP user name and password.
16#00000214	An account to save files is required.	Check the FTP user access rights.
16#00000226	Execution of the requested file operation was not possible because using the file was not possible, e.g., accessing it was not possible because it was not found.	Make sure that a file with the specified name exists in the directory on the FTP server. Check the access rights of the specified file.
16#00000229	Execution was not possible because the file name was not correct.	Check the access rights of the specified directory.

Version Information

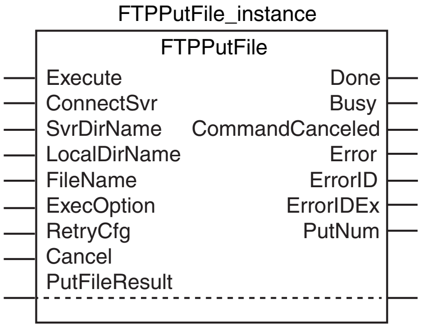
A CPU Unit with unit version 1.08 or later and Sysmac Studio version 1.09 or higher are required to use this instruction.

Sample Programming

Refer to the sample programming for the FTPGetFileList instruction (page 2-1111).

FTPPutFile

The FTPPutFile instruction uploads a file to the FTP server.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FTPPutFile	Put File onto FTP Server	FB		FTPPutFile_instance(Execute, ConnectSvr, SvrDirName, LocalDirName, FileName, ExecOption, RetryCfg, Cancel, PutFileResult, Done, Busy, CommandCanceled, Error, ErrorID, ErrorIDEx, PutNum) ;

Variables

<table>
<tr><th>Name</th><th>Meaning</th><th>I/O</th><th>Description</th><th>Valid range</th><th>Unit</th><th>Default</th></tr>
<tr><td>Connect Svr</td><td>Connected FTP server settings</td><td rowspan="7">Input</td><td>Setting parameters for the connected FTP server</td><td>---</td><td rowspan="7">---</td><td>*1</td></tr>
<tr><td>SvrDir Name</td><td>FTP server directory name</td><td>Name of FTP server directory to which to upload a file</td><td>256 bytes max. (255 single-byte alphanumeric characters plus the final NULL character)*2</td><td>''*3</td></tr>
<tr><td>LocalDir Name</td><td>Local directory name</td><td>Name of the directory in which to store the file uploaded to the FTP server</td><td rowspan="2">256 bytes max. (255 single-byte alphanumeric characters plus the final NULL character)</td><td>'/'</td></tr>
<tr><td>FileName</td><td>File name</td><td>Name of file to upload*4</td><td>*1</td></tr>
<tr><td>Exec Option</td><td>FTP execution options</td><td>Options for FTP execution</td><td rowspan="2">---</td><td rowspan="2">---</td></tr>
<tr><td>RetryCfg</td><td>Execution retry settings</td><td>Instruction execution retry settings</td></tr>
<tr><td>Cancel</td><td>Cancel</td><td>TRUE: Instruction execution is canceled.
FALSE: Instruction execution is not canceled.</td><td>Depends on data type.</td><td>FALSE</td></tr>
</table>

Name	Meaning	I/O	Description	Valid range	Unit	Default
PutFile Result[] array*5*6*7	Uploaded file results	In-out	Uploaded file results	---	---	*1
CommandC anceled	Cancel completed	Output	TRUE: Canceling completed. FALSE: Canceling not completed.	Depends on data type.	---	---
PutNum	Number of files to upload		Number of files to upload	---		

*1 If you omit an input parameter, the default value is not applied. A building error will occur.

*2 You cannot use the following characters in FTP server directory names: * ? < > | "

*3 The default is the home directory when you log onto the FTP server.

*4 You can use wildcards in file names.

*5 The array can have a maximum of 1,000 elements.

*6 This is a one-dimensional array. If an array with more than one dimension is specified, a building error will occur.

*7 The first array element number is 0. If a number other than 0 is specified for the first array element, a building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
ConnectSvr	Refer to *Function* for details on the structure _sFTP_CONNECT_SVR.																			
SvrDirName																				OK
LocalDirName																				OK
FileName																				OK
ExecOption	Refer to *Function* for details on the structure _sFTP_EXEC_OPTION.																			
RetryCfg	Refer to *Function* for details on the structure _sFTP_RETRY_CFG.																			
Cancel	OK																			
PutFileResult[] array	Refer to *Function* for details on the structure _sFTP_FILE_RESULT.																			
Command Canceled	OK																			
PutNum							OK													

Function

The FTPPutFile instruction uploads the file specified with *FileName* in the specified directory *LocalDirName* in the SD Memory Card to the directory specified with *SvrDirName* on the connected FTP server *ConnectSvr*. If the specified directory *SvrDirName* does not exist on the FTP server, a new directory is created and the specified file is uploaded.

You can use wildcards in *FileName*. This allows you to upload more than one file at one time.

The results of uploading is stored in *PutFileResult[]* for each file. Store the number of files to upload in *PutNum*. If you use a wildcard in *FileName*, store the number of files with names that match the wildcard.

If the actual number of transferred files is different, the value of *PutFileResult[].TxError* changes to TRUE.

If an error occurs when deleting the source file after the upload, the value of *PutFileResult[].RemoveError* changes to TRUE.

With an NY-series Controller, files are downloaded into the shared folder (Virtual SD Memory Card). Before downloading files to the Virtual SD Memory Card, you must make the settings for the Virtual SD Memory Card. Refer to the *NY-series Industrial Panel PC / Industrial Box PC Software User's Manual* (Cat. No. W558) for details on a Virtual SD Memory Card.

The data type of *ConnectSvr* is structure _sFTP_CONNECT_SVR. The specifications are as follows:

Name		Meaning	Description	Data type	Valid range	Unit	Default
ConnectSvr		Connected FTP server settings	Setting parameters for the connected FTP server	_sFTP_CONNECT_SVR	---	---	---
	Adr	Address	IP address or host name*1	STRING	1 to 200 bytes*2	---	---
	PortNo	Port number	TCP port number of FTP server control connection	UINT	0 to 65535*3		
	UserName	User name	User name on FTP server	STRING	33 bytes max.*4*5*6		
	Password	Password	FTP server password	STRING	33 bytes max.*4*5*6		

*1 A separate DNS or Hosts setting is required to specify a host name.

*2 You can use the following single-byte characters: A to Z, a to z, 0 to 9, - (hyphen), . (period), and _ (underbar).

*3 If you specify 0, TCP port number 21 is used.

*4 You can use the following single-byte characters: A to Z, a to z, 0 to 9, - (hyphen), . (period), and _ (underbar). You can also use \ and @ for a CPU Unit with unit version 1.16 or later.

*5 The NULL character at the end must be counted in the number of bytes.

*6 For CPU Units with unit version 1.08, specify a text string of one character or more. An error will occur if you specify a text string that contains only the final NULL character.

The data type of *PutFileResult[]* is structure _sFTP_FILE_RESULT. The specifications are as follows:

Name		Meaning	Description	Data type	Valid range	Unit	Default
PutFileResult		Uploaded file results	Transferred file results	_sFTP_FILE_RESULT	---	---	---
	Name	File name*1	Transferred file name	STRING	256 bytes max. (255 single-byte alphanumeric characters plus the final NULL character)	---	---
	TxError	Transfer error	TRUE: Transfer ended in an error. FALSE: Transfer ended normally.	BOOL	Depends on data type.		
	RemoveError	Deletion error	TRUE: Deletion ended in an error. FALSE: Deletion ended normally.	BOOL			
	Reserved	Reserved	Reserved by the system.	ARRAY[0..3] Of Byte	---		0

*1 The file name extension is included.

Using Wildcards to Specify File Names

You can use wildcards to specify the names of the files to upload.

Wildcard specifications are the same as those for the FTPGetFile instruction (page 2-1128). Refer to the specified page for details.

Specifying Options for FTP Server Processing

You can specify FTP server processing options when you upload files.

The option settings are the same as those for the FTPGetFile instruction (page 2-1128). Refer to the specified page for details.

Specifying Retrying Connection Processing with the FTP Server

You can specify retrying connection processing with the FTP server.

The operation for the retry settings is the same as that for the FTPGetFileList instruction (page 2-1111). Refer to the specified page for details.

Canceling Instruction Execution

You can cancel execution of the FTPPutFile instruction after execution has started.

The results of uploading files from the FTP server up to the point where it is canceled are stored in *PutNum* and *PutFileResult[]*.

The operation for cancellation is the same as that for the FTPGetFileList instruction (page 2-1111). Refer to the specified page for details.

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta[*1]	Online	BOOL	This variable indicates when built-in EtherNet/IP port communications can be used. TRUE: Communications are possible. FALSE: Communications are not possible.
_EIP1_EtnOnlineSta[*2] _EIP2_EtnOnlineSta[*3]			
_EIPIn1_EtnOnlineSta[*4]			
_Card1Ready	SD Memory Card Ready Flag	BOOL	This variable indicates whether the SD Memory Card is recognized and usable. TRUE : Can be used. FALSE: Cannot be used.

*1 Use this variable name for an NJ-series CPU Unit.

*2 Use this variable name for port 1 on an NX-series CPU Unit, or for an NY-series Controller.

*3 Use this variable name for port 2 on an NX-series CPU Unit.

*4 Use this variable name for the internal communication port on an NY-series Controller.

Precautions for Correct Use

- This instruction can be used only for the built-in EtherNet/IP ports on NJ/NX-series CPU Units and NY-series Controllers.
- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to Using this Section (page 2-3) for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- If the number of uploaded file results to store exceeds the number of array elements in *PutFileResult[],* the results that will not fit are not stored. In this case, *Error* does not change to TRUE.
- If a file name exceeds 255 characters, the first 255 characters are stored in *Name* in *PutFileResult[]*. In this case, *Error* does not change to TRUE.
- You can execute a maximum of 3 of the following instructions at the same time: FTPGetFileList, FTPGetFile, FTPPutFile, FTPRemoveFile, and FTPRemoveDir.
- If a wildcard is used in the file name and an error occurs for more than one file, the results of the first file for which the value of *PutFileResult[].TxError* is TRUE of all the files for which results are stored in *PutFileResult[]* are stored in *ErrorID* and *ErrorIDEx*.
- File names are not case sensitive. Therefore, if the only difference between the names of the files at the transfer destination and the transfer files is in capitalization, the files are detected as having the same names. The following is performed in this case.

Value of *OverWrite*	Overwrite specification	Processing
TRUE	Overwrite	If overwriting is not specified, the operation depends on the FTP server specifications.
FALSE	Do not overwrite.	The transfer destination files are not overwritten. The files are not transferred to the transfer destination.

- If the file specified by *FileName* does not exist in the specified directory on the SD Memory Card, a transfer error occurs and the value of *PutFileResult[].TxError* changes to TRUE.
- If the name specified for *FileName* is actually the name of a directory, a transfer error occurs and the value of *PutFileResult[].TxError* changes to TRUE.
- If *ExecOption.FileRemove* is TRUE and the file specified with *FileName* has a read-only attribute, a deletion error occurs and the value of *PutFileResult[].RemoveError* changes to TRUE.

- An error will occur in the following cases. *Error* will change to TRUE.
  - The value of any input parameter is outside of the valid range.
  - ".." is specified for a directory level in *SvrDirName* or *LocalDirName*.
  - An incorrect path such as "//" is specified for *SvrDirName* or *LocalDirName*.
  - The directory specified by *SvrDirName* does not exist on the FTP server.
  - The directory specified by *LocalDirName* does not exist on the FTP client.
  - More than 1,000 files to upload exist in the directory specified with *LocalDirName*.
  - The file directory specified with *FileName* does not exist in the upload source directory on the SD Memory Card.
  - *ExecOption.OverWrite* is FALSE and a file with the same name as the specified file name *FileName* already exists in the specified directory *SvrDirName*.
  - *ExecOption.FileRemove* is TRUE but a file with a name that matches *FileName* has a read-only attribute.
  - The FTP server specified by *ConnectSvr* does not exist on the network or the specified FTP server is not operating.
  - Accessing the file specified with *FileName* failed because there is no access right to the file or the file is corrupted.
  - More than 3 of the following instructions were executed at the same time: FTPGetFileList, FTPGetFile, FTPPutFile, FTPRemoveFile, and FTPRemoveDir.
  - The SD Memory Card is not in a usable condition.
- For this instruction, expansion error code *ErrorIDEx* gives the FTP response code that was returned by the FTP server. The following table lists typical values of *ErrorIDEx* and describes the meanings of the errors and the corrections. For details, refer to FTP server specifications. An expansion error code is output to *ErrorIDEx* when the value of error code *ErrorID* is WORD#16#2407.

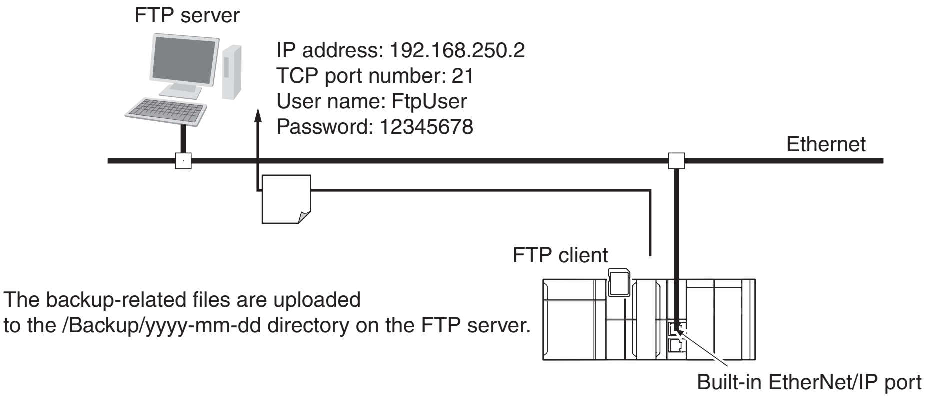
Value of *ErrorIDEx*	Meaning	Correction
16#000001A9	It was not possible to establish a data connection.	If you use FTP communications with an FTP server over the Internet, make sure that the FTP open mode is not set to active.
16#000001AA	The connection was closed. Data transfer was aborted.	Check the connection to the FTP server. Make sure that the FTP server is operating.
16#000001C2	It was not possible to perform the requested file operation. Using the file was not possible, e.g., it is already open.	Make sure that the file is not open for any other application.
16#00000212	User login was not possible.	Check the FTP user name and password.
16#00000214	An account to save files is required.	Check the FTP user access rights.
16#00000226	Execution of the requested file operation was not possible because using the file was not possible, e.g., accessing it was not possible because it was not found.	Make sure that a file with the specified name exists in the directory on the FTP server. Check the access rights of the specified file.
16#00000229	Execution was not possible because the file name was not correct.	Check the access rights of the specified directory.

Version Information

A CPU Unit with unit version 1.08 or later and Sysmac Studio version 1.09 or higher are required to use this instruction.

Sample Programming

This programming executes an SD Memory Card backup and then uploads all of the backup-related files to the /Backup/yyyy-mm-dd directory on the FTP server.



The Controller is connected to the FTP server through an EtherNet/IP network. The settings of the parameters to connect to the FTP server are given in the following table.

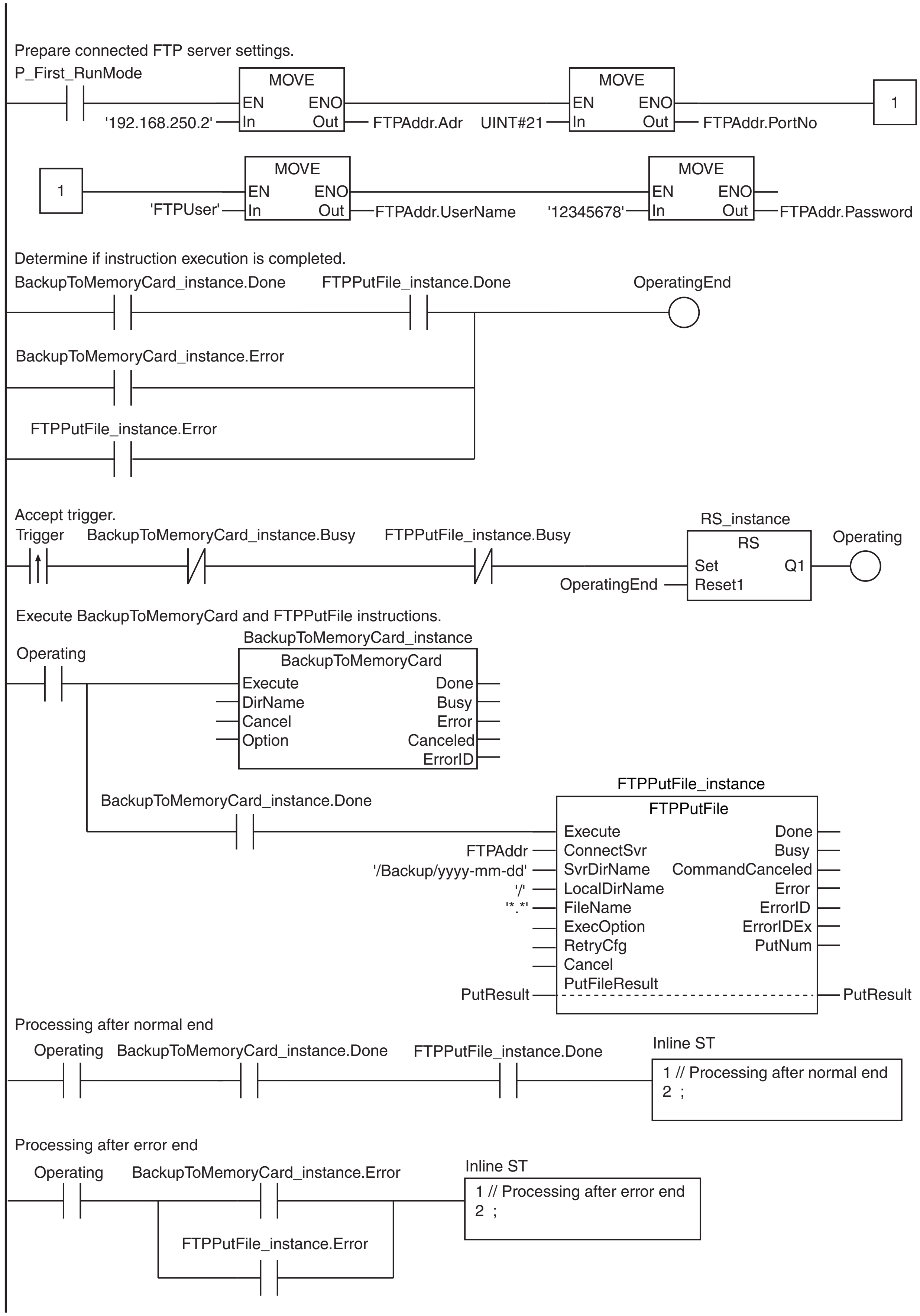
Parameter	Value
IP address	192.168.250.2
TCP port number	21
User name	FtpUser
Password	12345678

The following procedure is used.

1. The BackupToMemoryCard instruction is used to save NJ/NX-series Controller backup-related files to the root directory on the SD Memory Card.
2. The FTPPutFile instruction is used to upload the backup-related files to the /Backup/yyyy-mm-dd directory on the FTP server.
   The wildcard specification *.* is used to specify the names of the files to transfer.
3. Normal end processing is executed if all processing ends normally. Processing for an error end is performed if an error occurs.

LD

Internal Variables	Variable	Data type	Initial value	Comment
	FTPPutFile_instance	FTPPutFile		Instance of FTPPutFile instruction
	FTPAddr	_sFTP_CONNECT_SVR	(Adr := '', PortNo := 0, UserName := '', Password := '')	Connected FTP server settings
	PutResult	ARRAY[0..0] OF _sFTP_FILE_RESULT	[(Name := '', TxError := False, RemoveError := False, Reserved := [4(16#0)])]	Uploaded file results
	RS_instance	RS		Instance of RS instruction
	OperatingEnd	BOOL	FALSE	Processing completed
	Trigger	BOOL	FALSE	Execution condition
	Operating	BOOL	FALSE	Processing
	BackupToMemoryCard_instance	BackupToMemoryCard		Instance of BackupToMemoryCard instruction


Prepare connected FTP server settings.
P_First_RunMode
MOVE
EN ENO
In Out
'192.168.250.2'
FTPAddr.Adr
UINT#21
FTPAddr.PortNo
1
'FTPUser'
FTPAddr.UserName
'12345678'
FTPAddr.Password
Determine if instruction execution is completed.
BackupToMemoryCard_instance.Done
FTPPutFile_instance.Done
OperatingEnd
BackupToMemoryCard_instance.Error
FTPPutFile_instance.Error
Accept trigger.
Trigger
BackupToMemoryCard_instance.Busy
FTPPutFile_instance.Busy
RS_instance
RS
Set Q1
Reset1
Operating
OperatingEnd
Execute BackupToMemoryCard and FTPPutFile instructions.
BackupToMemoryCard_instance
BackupToMemoryCard
Operating
Execute Done
DirName Busy
Cancel Error
Option Canceled
ErrorID
FTPPutFile_instance
FTPPutFile
BackupToMemoryCard_instance.Done
Execute Done
FTPAddr
ConnectSvr Busy
'/Backup/yyyy-mm-dd'
SvrDirName CommandCanceled
'/'
LocalDirName Error
'*.*'
FileName ErrorID
ExecOption ErrorIDEx
RetryCfg PutNum
Cancel
PutFileResult
PutResult
PutResult
Processing after normal end
Operating
BackupToMemoryCard_instance.Done
FTPPutFile_instance.Done
Inline ST
1 // Processing after normal end
2 ;
Processing after error end
Operating
BackupToMemoryCard_instance.Error
FTPPutFile_instance.Error
Inline ST
1 // Processing after error end
2 ;

ST

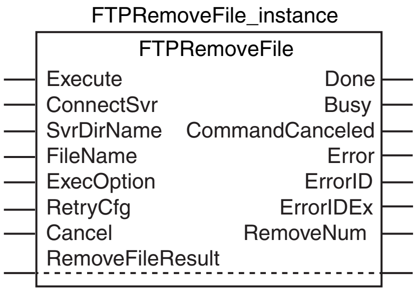
Internal Vari-ables	Variable	Data type	Initial value	Comment
	R_TRIG_instance	R_TRIG		Instance of R_TRIG instruction
	UP_Q	BOOL	FALSE	Trigger output
	FTPPutFile_in-stance	FTPPutFile		Instance of FTPPutFile instruction
	DoFTPTrigger	BOOL	FALSE	Execution condition for BackupToMemoryCard and FTPPutFile
	FTPAddr	_sFTP_CON-NECT_SVR	(Adr := '', PortNo := 0, User-Name := '', Password := '')	Connected FTP server settings
	PutResult	ARRAY[0..0] OF _sFTP_FILE_RE-SULT	[(Name := '', TxError := False, RemoveError := False, Reserved := [4(16#0)])]	Uploaded file results
	Stage	UINT	0	Instruction execution stage
	Trigger	BOOL	FALSE	Execution condition
	BackupToMemo-ryCard_instance	BackupToMemory-Card		Instance of Backup-ToMemoryCard instruction

```
// Prepare connected FTP server settings.
IF P_First_RunMode THEN
   FTPAddr.Adr      := '192.168.250.2';// Address
   FTPAddr.PortNo   := UINT#21;        // Port number
   FTPAddr.UserName := 'FtpUser';      // User name
   FTPAddr.Password := '12345678';     // Password
END_IF;

// Accept trigger.
R_TRIG_instance(Trigger, UP_Q);
IF ( (UP_Q = TRUE) AND (BackupToMemoryCard_instance.Busy = FALSE) AND
     (FTPPutFile_instance.Busy = FALSE) ) THEN
   DoFTPTrigger := TRUE;
   Stage := INT#1;
   BackupToMemoryCard_instance(  // Initialize instance.
      Execute := FALSE) ;
   FTPPutFile_instance( // Initialize instance.
      Execute       := FALSE,
      ConnectSvr    := FTPAddr,
      SvrDirName    := '/Backup/yyyy-mm-dd',
      LocalDirName  := '/',
      FileName      := '*.*',
      PutFileResult := PutResult) ;
END_IF;

IF (DoFTPTrigger = TRUE) THEN
   CASE Stage OF
      1: // Execute BackupToMemoryCard instruction.
         BackupToMemoryCard_instance(
            Execute := TRUE,   // Execution
         IF (BackupToMemoryCard_instance.Done = TRUE) THEN
            Stage   := INT#2;  // To next stage
         ELSIF (BackupToMemoryCard_instance.Error = TRUE) THEN
            Stage   := INT#10; // Error end
         END_IF;
      2: // Execute FTPPutFile instruction.
```

```
        FTPPutFile_instance(
           Execute       := TRUE,                // Execution
           ConnectSvr    := FTPAddr,             // Connected FTP server
           SvrDirName    := '/Backup/yyyy-mm-dd',// FTP server directory name
           LocalDirName  := '/',                 // Local directory name
           FileName      := '*.*',               // File name
           PutFileResult := PutResult) ;         // Uploaded file results
        IF (FTPPutFile_instance.Done = TRUE) THEN
           Stage := INT#0;  // Normal end
        ELSIF (FTPPutFile_instance.Error = TRUE) THEN
           Stage := INT#20; // Error end
        END_IF;
     0:  // Processing after normal end
        DoFTPTrigger:=FALSE;
        Trigger     :=FALSE;
     ELSE // Processing after error end
        DoFTPTrigger:=FALSE;
        Trigger     :=FALSE;
  END_CASE;
END_IF;
```

FTPRemoveFile

The FTPRemoveFile instruction deletes a file from the FTP server.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FTPRemove File	Delete FTP Server File	FB		FTPRemoveFile_instance(Execute, ConnectSvr, SvrDirName, FileName, ExecOption, RetryCfg, Cancel, RemoveFileResult, Done, Busy, CommandCanceled, Error, ErrorID, ErrorIDEx, RemoveNum) ;

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Connect Svr	Connected FTP server settings		Setting parameters for the connected FTP server	---		*1
SvrDir Name	FTP server directory name		Name of FTP server directory containing the file to delete	256 bytes max. (255 single-byte alphanumeric characters plus the final NULL character)*2		''*3
FileName	File name	Input	Name of file to delete*4	256 bytes max. (255 single-byte alphanumeric characters plus the final NULL character)*5	---	*1
Exec Option	FTP execution options		Options for FTP execution	---		
RetryCfg	Execution retry settings		Instruction execution retry settings			---
Cancel	Cancel		TRUE: Instruction execution is canceled. FALSE: Instruction execution is not canceled.	Depends on data type.		

Name	Meaning	I/O	Description	Valid range	Unit	Default
Remove FileResult []array*6*7*8	Deleted file results	In-out	Deleted file results	---	---	*1
Command Canceled	Cancel completed	Output	TRUE: Canceling completed. FALSE: Canceling not completed.	Depends on data type.	---	---
Remove Num	Number of files to delete		Number of files to delete	---		

*1 If you omit an input parameter, the default value is not applied. A building error will occur.

*2 You cannot use the following characters in FTP server directory names: * ? < > | "

*3 The default is the home directory when you log onto the FTP server.

*4 You can use wildcards in file names.

*5 You cannot use the following character in file names: |

*6 The array can have a maximum of 1,000 elements.

*7 This is a one-dimensional array. If an array with more than one dimension is specified, a building error will occur.

*8 The first array element number is 0. If a number other than 0 is specified for the first array element, a building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
ConnectSvr	Refer to *Function* for details on the structure _sFTP_CONNECT_SVR.																			
SvrDirName																				OK
FileName																				OK
ExecOption	Refer to *Function* for details on the structure _sFTP_EXEC_OPTION.																			
RetryCfg	Refer to *Function* for details on the structure _sFTP_RETRY_CFG.																			
Cancel	OK																			
Remove FileResult[] array	Refer to *Function* for details on the structure _sFTP_FILE_RESULT.																			
Command Canceled	OK																			
RemoveNum							OK													

Function

The FTPRemoveFile instruction deletes the file specified by *FileName* in the specified directory *SvrDirName* on the connected FTP server *ConnectSvr*.

You can use wildcards in *FileName*. This allows you to delete more than one file at one time.

The results of deleting files is stored by file in *RemoveFileResult[]*. Store the number of files to delete in *RemoveNum*. If you use a wildcard in *FileName*, store the number of files with names that match the wildcard.

If the actual number of deleted files is different, the value of *RemoveFileResult[].RemoveError* changes to TRUE.

The data type of *ConnectSvr* is structure _sFTP_CONNECT_SVR. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
ConnectSvr	Connected FTP server settings	Setting parameters for the connected FTP server	_sFTP_CONNECT_SVR	---	---	---
Adr	Address	IP address or host name*1	STRING	1 to 200 bytes*2	---	---
PortNo	Port number	TCP port number of FTP server control connection	UINT	0 to 65535*3		
UserName	User name	User name on FTP server	STRING	33 bytes max.*4*5*6		
Password	Password	FTP server password	STRING	33 bytes max.*4*5*6		

*1 A separate DNS or Hosts setting is required to specify a host name.

*2 You can use the following single-byte characters: A to Z, a to z, 0 to 9, - (hyphen), . (period), and _ (underbar).

*3 If you specify 0, TCP port number 21 is used.

*4 You can use the following single-byte characters: A to Z, a to z, 0 to 9, - (hyphen), . (period), and _ (underbar). You can also use \ and @ for a CPU Unit with unit version 1.16 or later.

*5 The NULL character at the end must be counted in the number of bytes.

*6 For CPU Units with unit version 1.08, specify a text string of one character or more. An error will occur if you specify a text string that contains only the final NULL character.

The data type of *RemoveFileResult[]* is structure _sFTP_FILE_RESULT. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
RemoveFile Result	Deleted file results	Transferred file results	_sFTP_FILE_RESULT	---	---	---
Name	File name*1	Transferred file name	STRING	256 bytes max. (255 single-byte alphanumeric characters plus the final NULL character)	---	---
TxError	Transfer error	TRUE: Transfer ended in an error. FALSE: Transfer ended normally.	BOOL	Depends on data type.		
RemoveError	Deletion error	TRUE: Deletion ended in an error. FALSE: Deletion ended normally.	BOOL			
Reserved	Reserved	Reserved by the system.	ARRAY[0..3] Of Byte	---		0

*1 The file name extension is included.

Using Wildcards to Specify File Names

You can use wildcards to specify the names of the files to delete.

Wildcard specifications are the same as those for the FTPGetFile instruction (page 2-1128). Refer to the specified page for details.

Specifying Options for FTP Server Processing

The operation specified with *ExecOption* is performed to delete the files from the FTP server.

The option settings are the same as those for the FTPGetFile instruction (page 2-1128). Refer to the specified page for details.

However, the only option that is valid for this instruction is *ExecOption.PassiveMode*.

Specifying Retrying Connection Processing with the FTP Server

You can specify retrying connection processing with the FTP server.

The operation for the retry settings is the same as that for the FTPGetFileList instruction (page 2-1111). Refer to the specified page for details.

Canceling Instruction Execution

You can cancel execution of the FTPRemoveFile instruction after execution has started.

The results of deleting files from the FTP server up to the point where it is canceled are stored in *RemoveNum* and *RemoveFileResult[].*

The operation for cancellation is the same as that for the FTPGetFileList instruction (page 2-1111). Refer to the specified page for details.

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta*1	Online	BOOL	This variable indicates when built-in EtherNet/IP port communications can be used. TRUE: Communications are possible. FALSE: Communications are not possible.
_EIP1_EtnOnlineSta*2 _EIP2_EtnOnlineSta*3			
_EIPIn1_EtnOnlineSta*4			

*1 Use this variable name for an NJ-series CPU Unit.

*2 Use this variable name for port 1 on an NX-series CPU Unit, or for an NY-series Controller.

*3 Use this variable name for port 2 on an NX-series CPU Unit.

*4 Use this variable name for the internal communication port on an NY-series Controller.

Precautions for Correct Use

- This instruction can be used only for the built-in EtherNet/IP ports on the NJ/NX-series CPU Units and the NY-series Controllers.
- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to Using this Section (page 2-3) for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- If the number of deleted files exceeds the number of array elements in *RemoveFileResult[],* the results that will not fit are not stored. In this case, *Error* does not change to TRUE.
- If a file name exceeds 255 characters, the first 255 characters are stored in *Name* in *RemoveFileResult[]*. In this case, *Error* does not change to TRUE.
- You can execute a maximum of 3 of the following instructions at the same time: FTPGetFileList, FTPGetFile, FTPPutFile, FTPRemoveFile, and FTPRemoveDir.

- If a wildcard is used in the file name and an error occurs for more than one file, the results of the first file for which the value of *RemoveFileResult[].TxError* is TRUE of all the files for which results are stored in *RemoveFileResult[]* are stored in *ErrorID* and *ErrorIDEx*.
- In the following cases, the value of *RemoveFileResult[].RemoveError* changes to TRUE.
  - The file directory specified with *FileName* does not exist on the FTP server.
  - A file specified with *FileName* has a read-only attribute.
  - The name specified for *FileName* is actually the name of a directory.
- An error will occur in the following cases. *Error* will change to TRUE.
  - The value of any input parameter is outside of the valid range.
  - ".." is specified for a directory level in *SvrDirName*.
  - An incorrect path such as "//" is specified for *SvrDirName*.
  - The directory specified by *SvrDirName* does not exist on the FTP server.
  - More than 1,000 files to delete exist in the directory specified with *SvrDirName*.
  - A file that matches the file name specified with a wildcard in *FileName* does not exist in the directory on the FTP server.
  - A file specified with *FileName* has a read-only attribute.
  - The FTP server specified by *ConnectSvr* does not exist on the network or the specified FTP server is not operating.
  - More than 3 of the following instructions were executed at the same time: FTPGetFileList, FTPGetFile, FTPPutFile, FTPRemoveFile, and FTPRemoveDir.
- For this instruction, expansion error code *ErrorIDEx* gives the FTP response code that was returned by the FTP server. The following table lists typical values of *ErrorIDEx* and describes the meanings of the errors and the corrections. For details, refer to FTP server specifications. An expansion error code is output to *ErrorIDEx* when the value of error code *ErrorID* is WORD#16#2407.

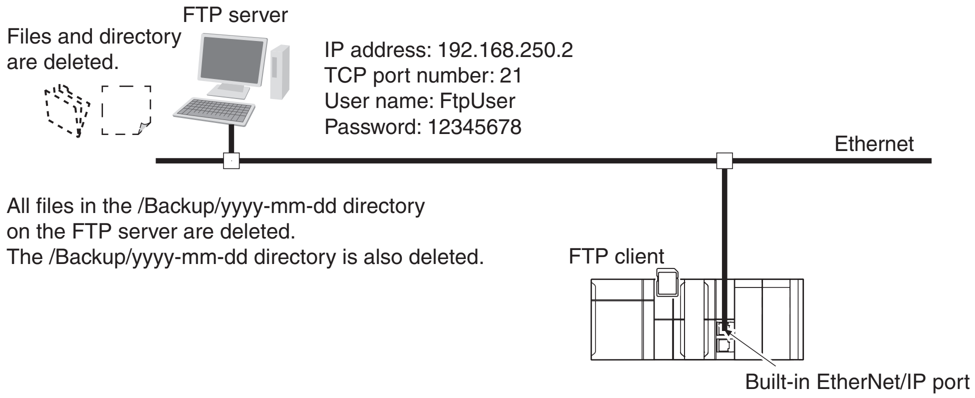
Value of *ErrorIDEx*	Meaning	Correction
16#000001A9	It was not possible to establish a data connection.	If you use FTP communications with an FTP server over the Internet, make sure that the FTP open mode is not set to active.
16#000001AA	The connection was closed. Data transfer was aborted.	Check the connection to the FTP server. Make sure that the FTP server is operating.
16#000001C2	It was not possible to perform the requested file operation. Using the file was not possible, e.g., it is already open.	Make sure that the file is not open for any other application.
16#00000212	User login was not possible.	Check the FTP user name and password.
16#00000214	An account to save files is required.	Check the FTP user access rights.
16#00000226	Execution of the requested file operation was not possible because using the file was not possible, e.g., accessing it was not possible because it was not found.	Make sure that a file with the specified name exists in the directory on the FTP server. Check the access rights of the specified file.
16#00000229	Execution was not possible because the file name was not correct.	Check the access rights of the specified directory.

Version Information

A CPU Unit with unit version 1.08 or later and Sysmac Studio version 1.09 or higher are required to use this instruction.

Sample Programming

This programming deletes all of the files in the /Backup/yyyy-mm-dd directory on the FTP server. It then deletes the /Backup/yyyy-mm-dd directory too.



The Controller is connected to the FTP server through an EtherNet/IP network. The settings of the parameters to connect to the FTP server are given in the following table.

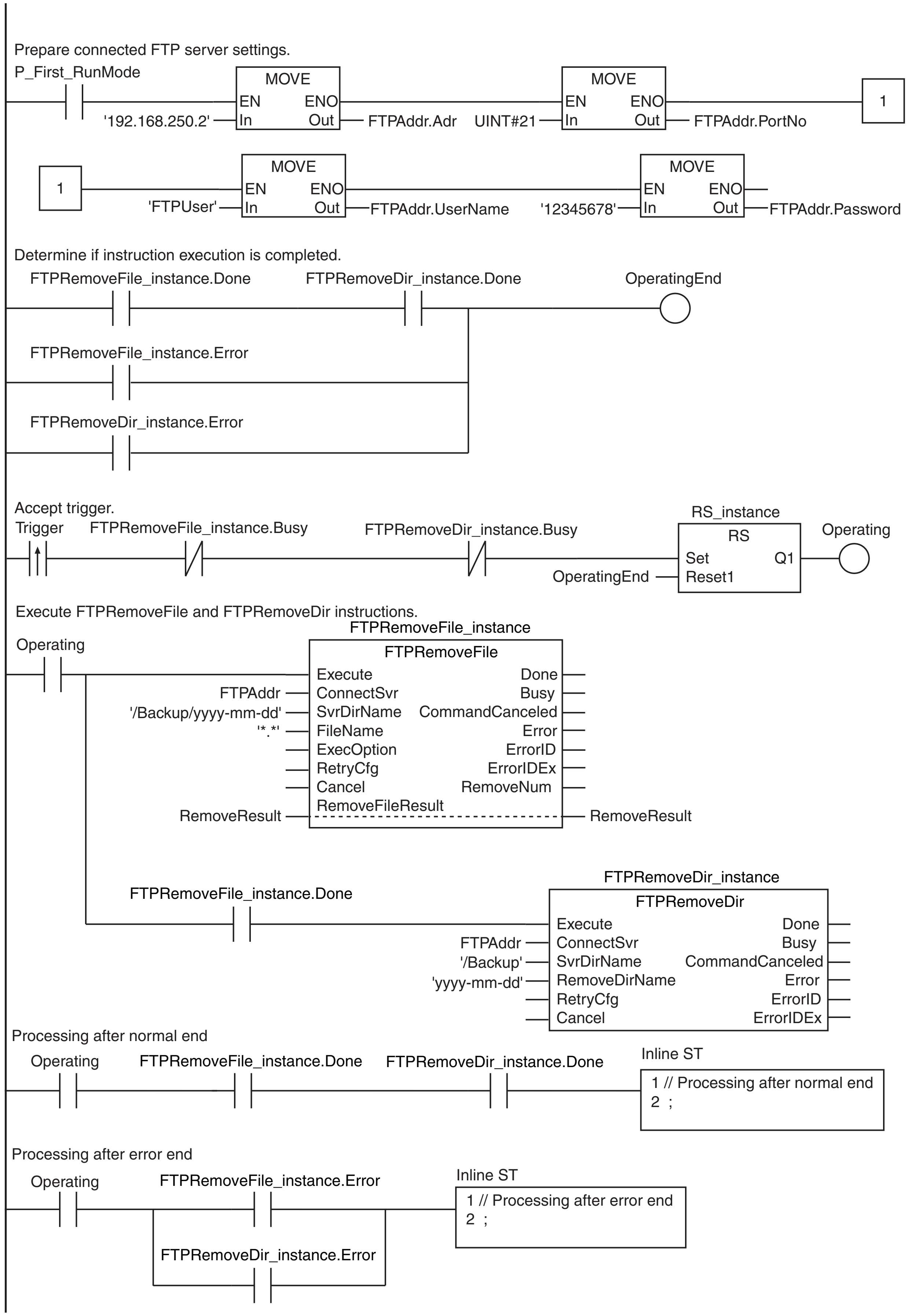
Parameter	Value
IP address	192.168.250.2
TCP port number	21
User name	FtpUser
Password	12345678

The following procedure is used.

1. The FTPRemoveFile instruction is used to delete all of the files in the /Backup/yyyy-mm-dd directory on the FTP server. The wildcard specification *.* is used to specify the names of the files to delete.
2. The FTPRemoveDir instruction is used to delete the /Backup/yyyy-mm-dd directory from the FTP server.
3. Normal end processing is executed if all processing ends normally. Processing for an error end is performed if an error occurs.

LD

Internal Variables	Variable	Data type	Initial value	Comment
	FTPRemoveFile_instance	FTPRemoveFile		Instance of FTPRemoveFile instruction
	FTPRemoveDir_instance	FTPRemoveDir		Instance of FTPRemoveDir instruction
	FTPAddr	_sFTP_CONNECT_SVR	(Adr := '', PortNo := 0, UserName := '', Password := '')	Connected FTP server settings
	RemoveResult	ARRAY[0..0] OF _sFTP_FILE_RESULT	[(Name := '', TxError := False, RemoveError := False, Reserved := [4(16#0)])]	Deleted file results
	RS_instance	RS		Instance of RS instruction
	OperatingEnd	BOOL	FALSE	Processing completed
	Trigger	BOOL	FALSE	Execution condition
	Operating	BOOL	FALSE	Processing


Prepare connected FTP server settings.
P_First_RunMode
MOVE
EN ENO
In Out
'192.168.250.2'
FTPAddr.Adr
UINT#21
MOVE
EN ENO
In Out
FTPAddr.PortNo
1
1
MOVE
EN ENO
In Out
'FTPUser'
FTPAddr.UserName
'12345678'
MOVE
EN ENO
In Out
FTPAddr.Password
Determine if instruction execution is completed.
FTPRemoveFile_instance.Done
FTPRemoveDir_instance.Done
OperatingEnd
FTPRemoveFile_instance.Error
FTPRemoveDir_instance.Error
Accept trigger.
Trigger
FTPRemoveFile_instance.Busy
FTPRemoveDir_instance.Busy
RS_instance
RS
Set Q1
Reset1
OperatingEnd
Operating
Execute FTPRemoveFile and FTPRemoveDir instructions.
FTPRemoveFile_instance
Operating
FTPRemoveFile
Execute Done
FTPAddr ConnectSvr Busy
'/Backup/yyyy-mm-dd' SvrDirName CommandCanceled
'*.*' FileName Error
ExecOption ErrorID
RetryCfg ErrorIDEx
Cancel RemoveNum
RemoveResult RemoveFileResult RemoveResult
FTPRemoveDir_instance
FTPRemoveFile_instance.Done
FTPRemoveDir
Execute Done
FTPAddr ConnectSvr Busy
'/Backup' SvrDirName CommandCanceled
'yyyy-mm-dd' RemoveDirName Error
RetryCfg ErrorID
Cancel ErrorIDEx
Processing after normal end
Operating
FTPRemoveFile_instance.Done
FTPRemoveDir_instance.Done
Inline ST
1 // Processing after normal end
2 ;
Processing after error end
Operating
FTPRemoveFile_instance.Error
FTPRemoveDir_instance.Error
Inline ST
1 // Processing after error end
2 ;

ST

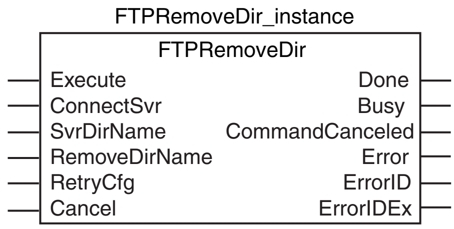
Internal Vari-ables	Variable	Data type	Initial value	Comment
	R_TRIG_instance	R_TRIG		Instance of R_TRIG instruction
	UP_Q	BOOL	FALSE	Trigger output
	FTPRemove-File_instance	FTPRemoveFile		Instance of FTPRemove-File instruction
	FTPRemove-Dir_instance	FTPRemoveDir		Instance of FTPRemove-Dir instruction
	DoFTPTrigger	BOOL	FALSE	Execution condition for FTPRemoveFile and FTPRemoveDir
	FTPAddr	_sFTP_CON-NECT_SVR	(Adr := '', PortNo := 0, User-Name := '', Password := '')	Connected FTP server settings
	RemoveResult	ARRAY[0..0] OF _sFTP_FILE_RE-SULT	[(Name := '', TxError := False, RemoveError := False, Reserved := [4(16#0)])]	Deleted file results
	Stage	UINT	0	Instruction execution stage
	Trigger	BOOL	FALSE	Execution condition

```
// Prepare connected FTP server settings.
IF P_First_RunMode THEN
   FTPAddr.Adr      := '192.168.250.2';// Address
   FTPAddr.PortNo   := UINT#21;        // Port number
   FTPAddr.UserName := 'FtpUser';      // User name
   FTPAddr.Password := '12345678';     // Password
END_IF;

// Accept trigger.
R_TRIG_instance(Trigger, UP_Q);
IF ( (UP_Q = TRUE) AND (FTPRemoveFile_instance.Busy = FALSE) AND
      (FTPRemoveDir_instance.Busy = FALSE) ) THEN
   DoFTPTrigger := TRUE;
   Stage := INT#1;
   FTPRemoveFile_instance( // Initialize instance.
      Execute          := FALSE,
      ConnectSvr       := FTPAddr,
      SvrDirName       := '/Backup/yyyy-mm-dd',
      FileName         := '*.*',
      RemoveFileResult := RemoveResult) ;
   FTPRemoveDir_instance( // Initialize instance.
      Execute       := FALSE,
      ConnectSvr    := FTPAddr,
      SvrDirName    := '/Backup',
      RemoveDirName := 'yyyy-mm-dd') ;
END_IF;

IF (DoFTPTrigger = TRUE) THEN
   CASE Stage OF
      1 : // Execute FTPRemoveFile instruction.
         FTPRemoveFile_instance(
            Execute          := TRUE,                 // Execution
            ConnectSvr       := FTPAddr,              // Connected FTP server
            SvrDirName       := '/Backup/yyyy-mm-dd', //FTP server directory name
            FileName         := '*.*',                // File name
            RemoveFileResult := RemoveResult) ;       // Deleted file results
```

```
            IF (FTPRemoveFile_instance.Done = TRUE) THEN
               Stage := INT#2;  // To next stage
            ELSIF (FTPRemoveFile_instance.Error = TRUE) THEN
               Stage := INT#10; // Error end
            END_IF;
         2 : // Execute FTPRemoveDir instruction.
            FTPRemoveDir_instance(
               Execute       := TRUE,          // Execution
               ConnectSvr    := FTPAddr,       // Connected FTP server
               SvrDirName    := '/Backup',     // FTP server directory name
               RemoveDirName := 'yyyy-mm-dd') ;// Directory to delete
            IF (FTPRemoveDir_instance.Done = TRUE) THEN
               Stage:=INT#0;  // Normal end
            ELSIF (FTPRemoveDir_instance.Error = TRUE) THEN
               Stage:=INT#20; // Error end
            END_IF;
         0 :  // Processing after normal end
            DoFTPTrigger:=FALSE;
            Trigger     :=FALSE;
         ELSE // Processing after error end
            DoFTPTrigger:=FALSE;
            Trigger     :=FALSE;
      END_CASE;
   END_IF;
```

FTPRemoveDir

The FTPRemoveDir instruction deletes a directory from the FTP server.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FTPRemove Dir	Delete FTP Server Directory	FB		FTPRemoveDir_instance(Execute, ConnectSvr, SvrDirName, RemoveDirName, Cancel, RetryCfg, Done, Busy, CommandCanceled, Error, ErrorID, ErrorIDEx) ;

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Connect Svr	Connected FTP server settings		Setting parameters for the connected FTP server	---		*1
SvrDir Name	FTP server directory name		Name of FTP server directory containing the directory to delete	256 bytes max. (255 single-byte alphanumeric characters plus the final NULL character)*2		''*3
RemoveDir-Name	Directory to delete	Input	Directory to delete	256 bytes max. (255 single-byte alphanumeric characters plus the final NULL character)	---	*1
RetryCfg	Execution retry settings		Instruction execution retry settings	---		---
Cancel	Cancel		TRUE: Instruction execution is canceled. FALSE: Instruction execution is not canceled.	Depends on data type.		FALSE
Command-Canceled	Cancel completed	Output	TRUE: Canceling completed. FALSE: Canceling not completed.	---	---	---

*1 If you omit an input parameter, the default value is not applied. A building error will occur.

*2 You cannot use the following characters in FTP server directory names: * ? < > | "

*3 The default is the home directory when you log onto the FTP server.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
ConnectSvr	Refer to *Function* for details on the structure _sFTP_CONNECT_SVR.																			
SvrDirName																				OK
RemoveDir Name																				OK
RetryCfg	Refer to *Function* for details on the structure _sFTP_RETRY_CFG.																			
Cancel	OK																			
Command Canceled	OK																			

Function

The FTPRemoveDir instruction deletes the specified directory *RemoveDirName* from the directory containing the directory to delete *SvrDirName* on the connected FTP server *ConnectSvr*.

When the value of *Done* in the instruction changes to TRUE, deletion of the specified directory is already completed. If the instruction fails to delete the directory, the value of *Error* changes to TRUE.

The data type of *ConnectSvr* is structure _sFTP_CONNECT_SVR. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
ConnectSvr	Connected FTP server settings	Setting parameters for the connected FTP server	_sFTP_CONNECT_SVR	---	---	---
Adr	Address	IP address or host name*1	STRING	1 to 200 bytes*2	---	---
PortNo	Port number	TCP port number of FTP server control connection	UINT	0 to 65535*3	---	---
UserName	User name	User name on FTP server	STRING	33 bytes max.*4*5*6	---	---
Password	Password	FTP server password	STRING	33 bytes max.*4*5*6	---	---

*1 A separate DNS or Hosts setting is required to specify a host name.

*2 You can use the following single-byte characters: A to Z, a to z, 0 to 9, - (hyphen), . (period), and _ (underbar).

*3 If you specify 0, TCP port number 21 is used.

*4 You can use the following single-byte characters: A to Z, a to z, 0 to 9, - (hyphen), . (period), and _ (underbar). You can also use \ and @ for a CPU Unit with unit version 1.16 or later.

*5 The NULL character at the end must be counted in the number of bytes.

*6 For CPU Units with unit version 1.08, specify a text string of one character or more. An error will occur if you specify a text string that contains only the final NULL character.

Specifying Retrying Connection Processing with the FTP Server

You can specify retrying connection processing with the FTP server.

The operation for retrying is the same as that for the FTPGetFileList instruction (page 2-1111). Refer to the specified page for details.

Canceling Instruction Execution

You can cancel execution of the FTPRemoveDir instruction after execution has started.

The operation for cancellation is the same as that for the FTPGetFileList instruction (page 2-1111). Refer to the specified page for details.

Related System-defined Variables

Name	Meaning	Data type	Description
_EIP_EtnOnlineSta*1	Online	BOOL	This variable indicates when built-in EtherNet/IP port communications can be used. TRUE: Communications are possible. FALSE: Communications are not possible.
_EIP1_EtnOnlineSta*2 _EIP2_EtnOnlineSta*3			
_EIPIn1_EtnOnlineSta*4			

*1 Use this variable name for an NJ-series CPU Unit.

*2 Use this variable name for port 1 on an NX-series CPU Unit, or for an NY-series Controller.

*3 Use this variable name for port 2 on an NX-series CPU Unit.

*4 Use this variable name for the internal communication port on an NY-series Controller.

Precautions for Correct Use

- This instruction can be used only for the built-in EtherNet/IP ports on the NJ/NX-series CPU Units and the NY-series Controllers.
- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to Using this Section (page 2-3) for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- Even if you use *Cancel* to cancel the execution of this instruction, sometimes the directory on the FTP server is deleted depending on the timing of when *Cancel* changes to TRUE.
  Check the directory on the FTP server.
- You can execute a maximum of 3 of the following instructions at the same time: FTPGetFileList, FTP-GetFile, FTPPutFile, FTPRemoveFile, and FTPRemoveDir.
- An error will occur in the following cases. *Error* will change to TRUE.
  - The value of any input parameter is outside of the valid range.
  - The directory specified by *SvrDirName* does not exist on the FTP server.
  - “..” is specified for a directory level in *SvrDirName* or *RemoveDirName*.
  - An incorrect path such as “//” is specified for *SvrDirName* or *RemoveDirName*.
  - The directory specified by *RemoveDirName* does not exist on the FTP server.
  - There are no files or subdirectories in the directory specified with *RemoveDirName*.
  - The directory specified with *RemoveDirName* has a read-only attribute.
  - The FTP server specified by *ConnectSvr* does not exist on the network or the specified FTP server is not operating.

- More than 3 of the following instructions were executed at the same time: FTPGetFileList, FTP-GetFile, FTPPutFile, FTPRemoveFile, and FTPRemoveDir.
- For this instruction, expansion error code *ErrorIDEx* gives the FTP response code that was returned by the FTP server. The following table lists typical values of *ErrorIDEx* and describes the meanings of the errors and the corrections. For details, refer to FTP server specifications. An expansion error code is output to *ErrorIDEx* when the value of error code *ErrorID* is WORD#16#2407.

Value of *ErrorI-DEx*	Meaning	Correction
16#000001A9	It was not possible to establish a data connection.	If you use FTP communications with an FTP server over the Internet, make sure that the FTP open mode is not set to active.
16#000001AA	The connection was closed. Data transfer was aborted.	Check the connection to the FTP server. Make sure that the FTP server is operating.
16#000001C2	It was not possible to perform the requested file operation. Using the file was not possible, e.g., it is already open.	Make sure that the file is not open for any other application.
16#00000212	User login was not possible.	Check the FTP user name and password.
16#00000214	An account to save files is required.	Check the FTP user access rights.
16#00000226	Execution of the requested file operation was not possible because using the file was not possible, e.g., accessing it was not possible because it was not found.	Make sure that a file with the specified name exists in the directory on the FTP server. Check the access rights of the specified file.
16#00000229	Execution was not possible because the file name was not correct.	Check the access rights of the specified directory.

Version Information

A CPU Unit with unit version 1.08 or later and Sysmac Studio version 1.09 or higher are required to use this instruction.

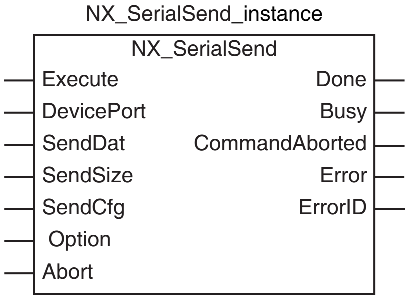
Sample Programming

Refer to the sample programming for the FTPRemoveFile instruction (page 2-1148).

Serial Communications Instructions

NX_SerialSend

The NX_SerialSend instruction sends data in No-protocol Mode from a serial port on an NX-series Communications Interface Unit or Option Board.

<table>
<tr><th>Instruction</th><th>Name</th><th>FB/ FUN</th><th>Graphic expression</th><th>ST expression</th></tr>
<tr><td>NX_SerialSend</td><td>Send No-protocol Data</td><td>FB</td><td>
</td><td>NX_SerialSend_instance(
Execute,
DevicePort,
SendDat,
SendSize,
SendCfg,
Option,
Abort,
Done,
Busy,
CommandAborted,
Error,
ErrorID);</td></tr>
</table>

Version Information

A CPU Unit with unit version 1.11 or later and Sysmac Studio version 1.15 or higher are required to use this instruction.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
DevicePort	Device port	Input	Object that represents a device port	---	---	---
SendDat[] (array)	Send data array		Send data array	Depends on data type.	---	*1
SendSize	Send data size		Send data size	0 to 4096	Bytes	0
SendCfg	Conditions attached to send data		Conditions attached to send data	---	---	---
Option	Option		Option	---	---	---
Abort	Interruption		Interruption of instruction execution	Depends on data type.	---	FALSE
Command-Aborted	Interruption completion	Output	Interruption completion	Depends on data type.	---	---

*1 If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
DevicePort	Refer to *Function* for details on the structure _sDEVICE_PORT.																			
SendDat[] (array)		OK																		
SendSize							OK													
SendCfg	Refer to *Function* for details on the structure _sSERIAL_CFG.																			
Option	Refer to *Function* for details on the structure _sSERIAL_SEND_OPTION.																			
Abort	OK																			
Command-Aborted	OK																			

Function

The NX_SerialSend instruction sends data in No-protocol Mode from the specified port on an NX-series Communications Interface Unit or Option Board.

The data type of the *DevicePort* input variable is structure _sDEVICE_PORT. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
DevicePort	Device port	Object that represents a device port	_sDEVICE_PORT	---	---	---
DeviceType	Device type	Type of the device to specify	_eDEVICE_TYPE	_DeviceNXUnit _DeviceEcat-Slave _DeviceOption-Board	---	---
NxUnit	Specified Unit	NX Unit to control	_sNXUNIT_ID	---	---	---
EcatSlave	Specified slave	EtherCAT slave to control	_sECAT_ID	---	---	---
OptBoard	Specified Option Board	Option Board to control	_sOPTBOARD_ID	---	---	---
Reserved	Reserved	Reserved	Reserved	---	---	---
PortNo	Port number	Port number 1: Port 1 2: Port 2	USINT	Depends on data type.	---	---

Use *DeviceType* to specify the device type. Set this to _DeviceNXUnit for an NX Unit and _DeviceOptionBoard for an Option Board. The variable used to specify the device is determined by the specified device type.

To specify an NX Unit, use *NxUnit* to specify the device.

In this case, *EcatSlave* and *OptBoard* are not used.

To *NxUnit*, pass the device variable that is assigned to the node location information on the I/O Map for the device to specify.

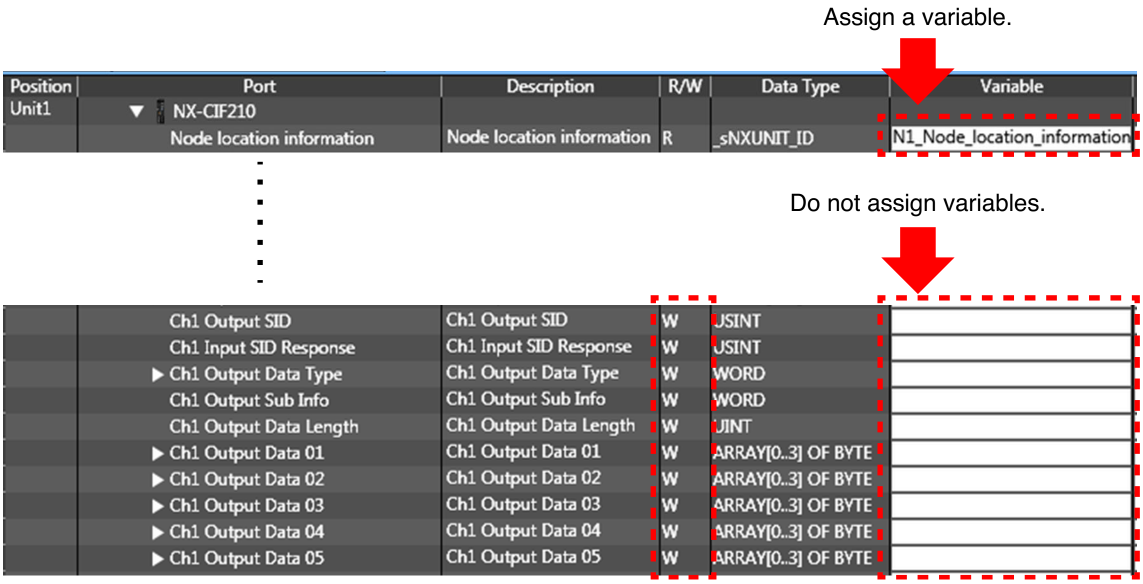
To specify an Option Board, use *OptBoard* to specify the device.

In this case, *NxUnit* and *EcatSlave* are not used.

To *OptBoard*, pass the device variable that is assigned to the node location information on the I/O Map for the device to specify.

If you use this instruction, be sure to assign a device variable to the node location information. Do not assign device variables to any I/O ports following the node location information that are indicated by “W” under the R/W column.

The figure below is an example of using this instruction for port 1 on an NX-CIF210.



Refer to the *Sysmac Studio Version 1 Operation Manual* (Cat. No. W504-E1-07 or later) for details on assigning a device variable to the node location information.

Use *PortNo* to specify the port number.

1: Port 1

2: Port 2

For an NX Unit, set this to Port 1 or Port 2.

For an Option Board, set this to Port 1.

The data type of *DeviceType* is enumerated type _eDEVICE_TYPE.

The meanings of the enumerators of enumerated type _eDEVICE_TYPE are as follows:

Enumerator	Meaning
_DeviceNXUnit	NX Unit is specified.
_DeviceEcatSlave	EtherCAT slave is specified.
_DeviceOptionBoard	Option Board is specified.

In this instruction, you can specify _DeviceNXUnit or _DeviceOptionBoard.

Data of the size specified with the *SendSize* input variable is sent from the send data specified with the *SendDat* input variable. If the value of *SendSize* is 0, nothing is sent. When the instruction is executed, the value of *Done* changes to TRUE instead of *Busy*.

To attach start and end codes to the send data, set them in the *SendCfg* input variable.

The data type of the *SendCfg* input variable is structure _sSERIAL_CFG. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
SendCfg	Conditions attached to send data	Conditions attached to send data	_sSERIAL_CFG	---	---	---
StartTrig	Start code existence	Start code existence	_eSERIAL_START	_SERIAL_START_NONE _SERIAL_START_STARTCODE1 _SERIAL_START_STARTCODE2	---	_SERIAL_START_NONE
StartCode	Start code	Start code	BYTE[2]	Depends on data type.	---	[2(16#0)]
EndTrig	End code existence	End code existence	_eSERIAL_END	_SERIAL_END_NONE _SERIAL_END_ENDCODE1 _SERIAL_END_ENDCODE2 _SERIAL_END_TERMINATION_CHAR _SERIAL_END_RCV_SIZE	---	_SERIAL_END_NONE
EndCode	End code	End code	BYTE[2]	Depends on data type.	---	[2(16#0)]
RcvSizeCfg	Receive size	Not used in this instruction.	UINT	0 to 4096	Bytes	0

The data type of *StartTrig* is enumerated type _eSERIAL_START.

The meanings of the enumerators of enumerated type _eSERIAL_START are as follows:

Enumerator	Meaning
_SERIAL_START_NONE	None
_SERIAL_START_STARTCODE1	1-byte code
_SERIAL_START_STARTCODE2	2-byte code

The data type of *EndTrig* is enumerated type _eSERIAL_END.

The meanings of the enumerators of enumerated type _sSERIAL_END are as follows:

Enumerator	Meaning
_SERIAL_END_NONE	None
_SERIAL_END_ENDCODE1	1-byte code
_SERIAL_END_ENDCODE2	2-byte code
_SERIAL_END_TERMINATION_CHAR	Termination condition
_SERIAL_END_RCV_SIZE	Receive size

Refer to *Operation of Start Code and End Code* on page 2-1181 for details on the operation of start code and end code.

To delay data transmission from the Controller to an NX-series Communications Interface Unit, set a delay time in units of 0.01 s with the *Option.SendDelay* input variable. The data type of the *Option* input variable is structure _sSERIAL_SEND_OPTION. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Option	Option	Option	_sSERIAL_SEND_OPTION	---	---	---
SendDelay	Send delay time	Send delay time	UINT	Depends on data type.	0.01 s	0

An error occurs if this instruction is executed for Units other than NX-series Communications Interface Units and Option Boards.

Operation of Start Code and End Code

Use *SendCfg.StartTrig* and *SendCfg.EndTrig* to specify the conditions of start and end codes that are attached to the send data.

If you attach a start or end code to the send data, exclude it from the value set for the *SendSize* input variable.

The operations of *StartTrig* and *EndTrig* are given below.

Value of *StartTrig*	Operation
_SERIAL_START_NONE	---
_SERIAL_START_STARTCODE1	*SendDat* is sent with start code attached to its beginning. Example: STX
_SERIAL_START_STARTCODE2	

Value of *EndTrig*	Operation
_SERIAL_END_NONE	---
_SERIAL_END_ENDCODE1	*SendDat* is sent with end code attached to its end. Example: ETX
_SERIAL_END_ENDCODE2	
_SERIAL_END_TERMINATION_CHAR	Error
_SERIAL_END_RCV_SIZE	Error

Interruption of Instruction Execution

If *Abort* is changed to TRUE during instruction execution, the execution is interrupted.

When the instruction execution is interrupted, *CommandAborted* changes to TRUE. The instruction is interrupted even when the data transmission is in progress.

If the change of *Abort* is too late to interrupt the execution, *Done* changes to TRUE and the instruction ends normally.

If both *Abort* and *Execute* are changed to TRUE, *CommandAborted* changes to TRUE.

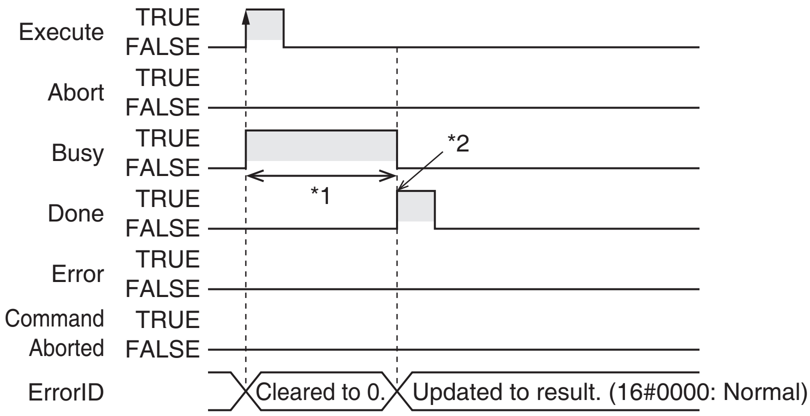
The interruption operation only finishes the *Busy* processing, and it does not clear the send buffer. To clear the buffer, use the NX_SerialBufClear instruction.

Timing Charts

The following figures show the timing charts.

● Normal end (when *SendDelay* is 0 (0 s))

The operation is as follows when *SendDelay* is 0 (0 s).

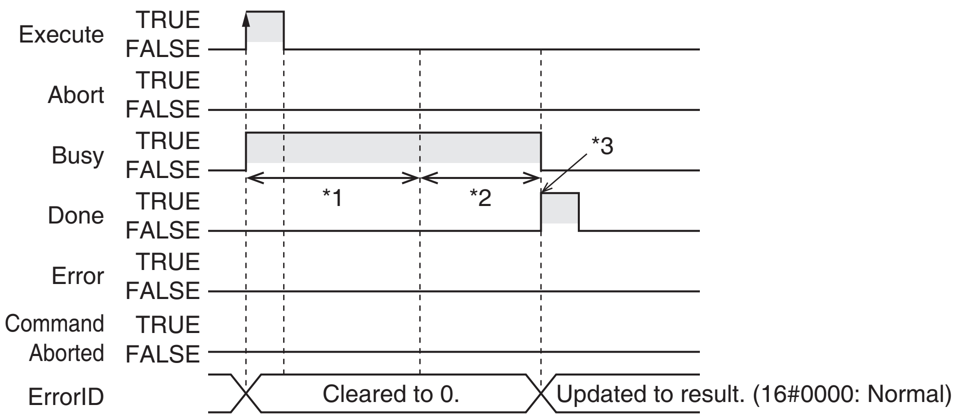


*1 Sending processing

*2 Sending completed

● Normal end (when *SendDelay* is 100 (1 s))

The operation is as follows when *SendDelay* is 100 (1 s).



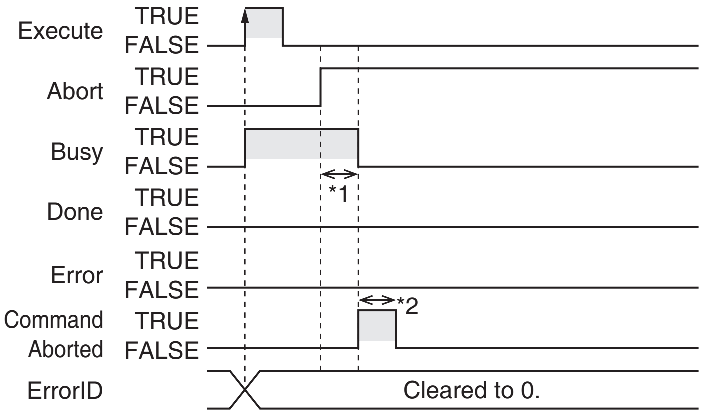
*1 The send delay time of 1 s

*2 Sending processing

*3 Sending completed

● Interruption executed (when *Busy* is TRUE)

The operation is as follows if *Abort* is changed to TRUE while *Busy* is TRUE.

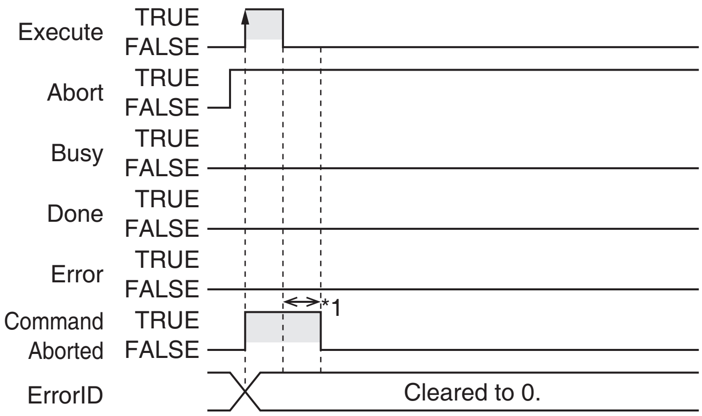


*1 Interruption processing

*2 Changes to FALSE after one task period.

● Interruption executed (when *Execute* is TRUE)

The operation is as follows if both *Abort* and *Execute* are changed to TRUE.



*1 Changes to FALSE after one task period.

Related System-defined Variables

The following device variable name is created automatically for an EtherCAT Coupler Unit whose device name is E001.

Name	Meaning	Data type	Description
_PLC_OptBoardSta	Option Board Status	ARRAY[1..2] of _sOPTBOARD_STA	• This stores the status of the Option Board.
_NXB_UnitIOActiveTbl	NX Unit I/O Data Active Status	ARRAY[0..8] OF BOOL	• This status tells the NX Units whether I/O data communications can be processed. • The subscript of the array corresponds to the NX Unit numbers. A subscript of 0 means the NX bus master.

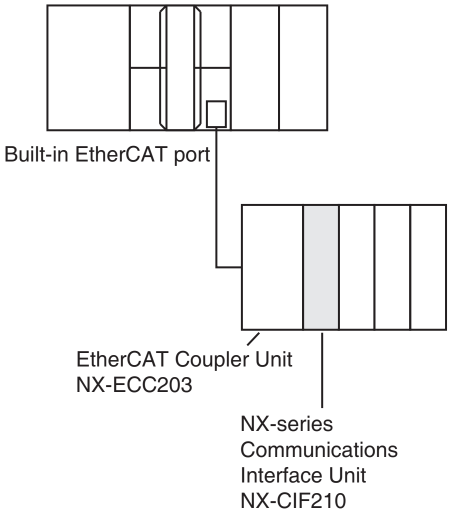
Precautions for Correct Use

- When *Abort* remains FALSE, this instruction is executed until the completion of processing even if *Execute* changes to FALSE or the execution time exceeds the task period.
  The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing. If *Abort* is changed to TRUE during instruction execution, *CommandAborted* or *Done* changes to TRUE.
- A compiling error will occur if you use this instruction in an event task. Do not use this instruction in event tasks.
- “CIF Unit Initialized” may occur when the NX-series Communications Interface Unit is restarted. Send or receive the data again, if necessary.
- If you use this instruction, do not assign device variables to any I/O ports that are indicated by “W” under the R/W column on the I/O Map Tab Page in the Sysmac Studio for the applicable NX-series Communications Interface Unit.
- An error will occur in the following cases. *Error* will change to TRUE.
  - A value that is out of range is set for *SendSize*, *SendCfg.StartTrig*, *SendCfg.EndTrig*, *DevicePort.DevicePortType*, or *DevicePort.PortNo*.
  - The array variable specified with *SendDat* is smaller than the size specified with *SendSize*.
  - The Unit, Option Board, or port specified with *DevicePort* does not exist.
  - The data type of *DevicePort* is invalid.
  - If more than 32 instructions from the NX_SerialSend instruction, NX_SerialRcv instruction, NX_ModbusRtuCmd instruction, NX_ModbusRtuRead instruction, NX_ModbusRtuWrite instruction, NX_SerialSigCtl instruction, NX_SerialSigRead instruction, NX_SerialStatusRead instruction, NX_SerialBufClear instruction, NX_SerialStartMon instruction and NX_SerialStopMon instruction are executed at the same time.
  - This instruction is executed with a device port variable that is the same as the one specified for the instruction which is still being executed. In this case, the instruction which is still being executed is one of the followings.
    The NX_SerialSend instruction, NX_ModbusRtuCmd instruction, NX_ModbusRtuRead instruction, and NX_ModbusWrite instruction.
  - This instruction is executed for Units other than NX-series Communications Interface Units and Option Boards.
  - The serial communications mode of the specified Option Board is not *No-protocol*.

Sample Programming

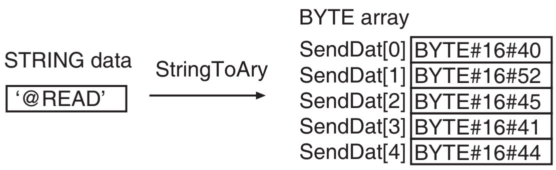
In this sample, an NX-series Communications Interface Unit (NX-CIF210) is connected to an EtherCAT Coupler Unit (NX-ECC203).

The unit number of the NX-CIF210 is set to 1.



A no-protocol command is sent to the barcode reader that is connected to serial port 2 of the NX-CIF210. The send command is the scene number acquisition command (@READ).

For the send command, the StringToAry instruction is used to separate the text string ‘@READ’ into individual characters and convert them to the character codes. The character codes are stored in the array elements of SendDat[].



There is no start code. End code is 16#OD (CR).

The settings of NX-CIF210 are given in the following table.

Item	Set value
Port 2: Baud Rate	38,400 bps
Port 2: Data Length	8 bits
Port 2: Parity	None
Port 2: Stop Bits	1 bit
Port 2: Flow Control	None

Definitions of Global Variables

Global Variables

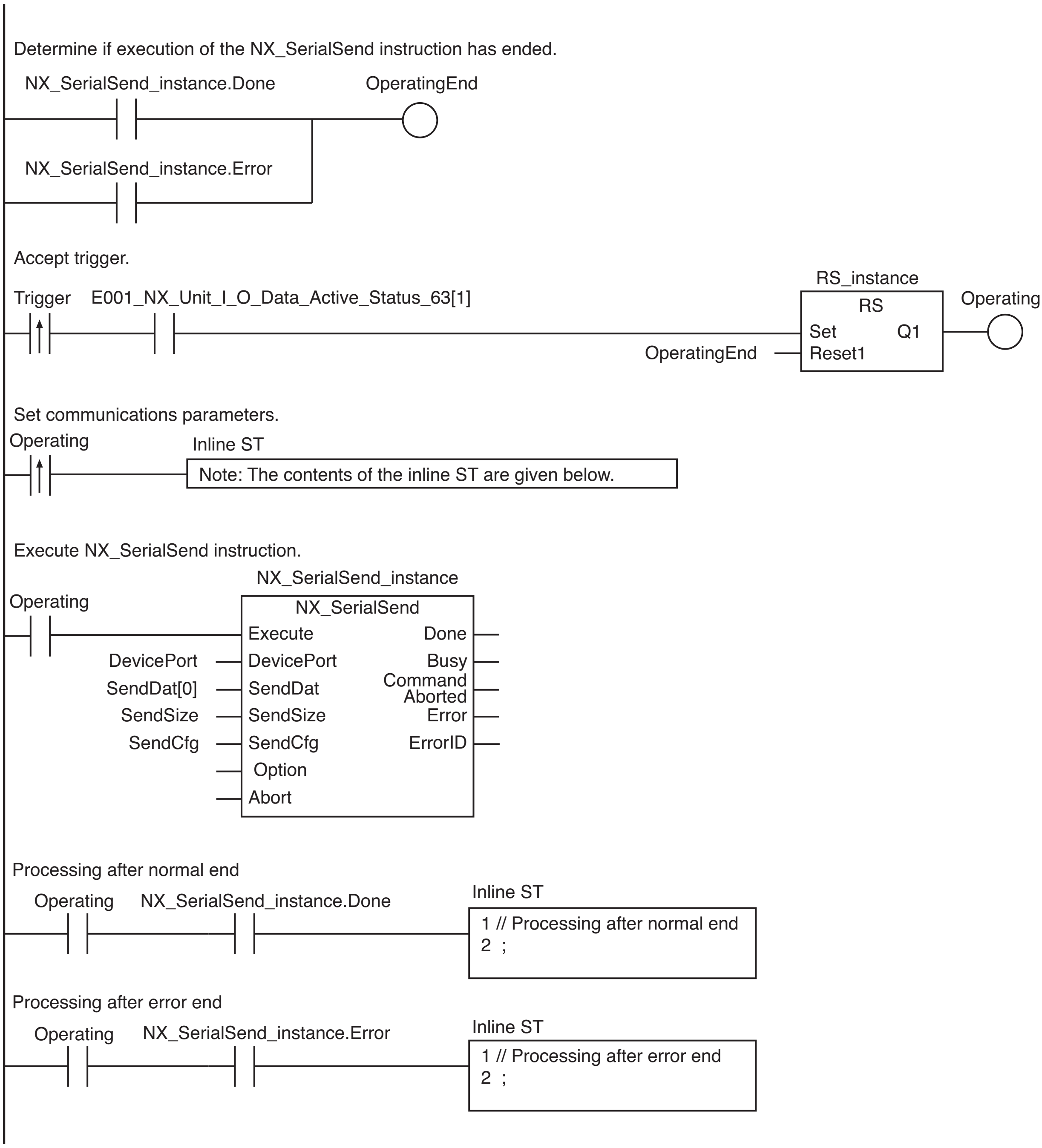
Name	Data type	AT	Comment
E001_NX_Unit_I_O_Data_Active_Status_63	ARRAY[0..63] OF BOOL	ECAT://node#1/NX Unit I/O Data Active Status 125	Usage of I/O data for 63 NX Units.
N1_Node_location_information	_sNXUNIT_ID	---	Device variable to specify NX-CIF210*1

*1 On the Sysmac Studio, right-click an NX-series slave terminal unit, select **Display Node Location Port**, and set the device variable. Refer to the *Sysmac Studio Version 1 Operation Manual* (Cat. No. W504-E1-07 or later) for details.

LD

Internal Variables	Variable	Data type	Initial value	Comment
	OperationEnd	BOOL	FALSE	Processing completed
	Trigger	BOOL	FALSE	Execution condition
	Operating	BOOL	FALSE	Processing
	DevicePort	_sDEVICE_PORT		Port settings
	SendDat	ARRAY [0..5] OF BYTE	[6(16#0)]	Send data
	SendSize	UINT	0	Send data size
	RS_instance	RS		
	NX_SerialSend_instance	NX_SerialSend		
	SendCfg	_sSERIAL_SEND_CFG		
	StartTrig	_eSERIAL_START	_SERIAL_START_NONE	Without start code
	StartCode	BYTE[2]	[2(16#0)]	
	EndTrig	_eSERIAL_END	_SERIAL_END_ENDCODE1	With end code
	EndCode	BYTE[2]	[16#0D,16#00]	16#0D(CR)

External Variables	Variable	Data type	Comment
	E001_NX_Unit_I_O_Data_Active_Status_63	ARRAY[0..63] OF BOOL	• Usage of I/O data for 63 NX Units. • If the relevant Unit number is 1, E001_NX_Unit_I_O_Data_Active_Status_63[1] is used.
	N1_Node_location_information	_sNXUNIT_ID	Device variable to specify NX-CIF210



● Contents of Inline ST

```
DevicePort.DeviceType:=_eDEVICE_TYPE#_DeviceNXUnit;
DevicePort.NxUnit:=N1_Node_location_information;
DevicePort.PortNo:=2;
StringToAry(In:='@READ', AryOut:=SendDat[0]);
SendSize := UINT#10#5;
```

ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	FALSE	Execution condition
	LastTrigger	BOOL	FALSE	Value of Trigger from previous task period
	OperatingStart	BOOL	FALSE	Processing started
	Operating	BOOL	FALSE	Processing
	DevicePort	_sDEVICE_PORT		Port settings
	SendDat	ARRAY [0..5] of BYTE	[6(16#0)]	Send data
	SendSize	UINT	0	Send data size
	NX_SerialSend_instance	NX_SerialSend		
	SendCfg	_sSERIAL_CFG		
	StartTrig	_eSERIAL_START	_SERIAL_START_NONE	Without start code
	StartCode	BYTE[2]	[2(16#0)]	
	EndTrig	_eSERIAL_END	_SERIAL_END_ENDCODE1	With end code
	EndCode	BYTE[2]	[16#0D,16#00]	16#0D(CR)

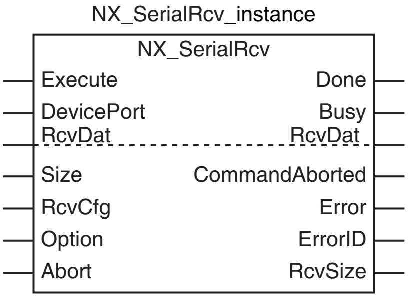
External Variables	Variable	Data type	Comment
	E001_NX_Unit_I_O_Data_Active_Status_63	ARRAY[0..63] OF BOOL	• Usage of I/O data for 63 NX Units. • If the relevant Unit number is 1, E001_NX_Unit_I_O_Data_Active_Status_63[1] is used.
	N1_Node_location_information	_sNXUNIT_ID	Device variable to specify NX-CIF210

```
// Detect when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE)
    AND (E001_NX_Unit_I_O_Data_Active_Status_63[1]) AND
(NX_SerialSend_instance.Busy=FALSE) ) THEN
        OperatingStart:=TRUE;
        Operating:=TRUE;
        DevicePort.DeviceType:=_eDEVICE_TYPE#_DeviceNXUnit;
        DevicePort.NxUnit:=N1_Node_location_information;
        DevicePort.PortNo:=2;
END_IF;
LastTrigger:=Trigger;

// Set communications parameters and initialize NX_SerialSend instruction.
IF (OperatingStart=TRUE) THEN
         NX_SerialSend_instance(
                             Execute:=FALSE,
                             DevicePort:=DevicePort;
                             SendDat:=SendDat[0],
                             SendSize:=UINT#1,
                             SendCfg:=SendCfg);
         StringToAry(In:='@READ', AryOut:=SendDat[0]);
         SendSize:=UINT#10#5;
         OperatingStart:=FALSE;
END_IF;

// Execute NX_SerialSend instruction.
```

```
IF (Operating=TRUE) THEN
        NX_SerialSend_instance(
                        Execute:=TRUE,
                        DevicePort:=DevicePort, // Port settings
                        SendDat:=SendDat[0],    // Send data
                        SendSize:=SendSize,     // Send data size
                        SendCfg:=SendCfg);      // End code settings
        IF (NX_SerialSend_instance.Done=TRUE) THEN
                // Processing after normal end

                Operating:=FALSE;
        END_IF;

        IF (NX_SerialSend_instance.Error=TRUE) THEN
                // Processing after error end

                Operating:=FALSE;
        END_IF;
END_IF;
```

NX_SerialRcv

The NX_SerialRcv instruction reads data in No-protocol Mode from a serial port on an NX-series Communications Interface Unit or Option Board.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
NX_SerialRcv	Receive No-protocol Data	FB		NX_SerialRcv_instance(Execute, DevicePort, RcvDat, Size, RcvCfg, Option, Abort, Done, Busy, CommandAborted, Error, ErrorID, RcvSize);

Version Information

A CPU Unit with unit version 1.11 or later and Sysmac Studio version 1.15 or higher are required to use this instruction.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
DevicePort	Device port	Input	Object that represents a device port	---	---	---
Size	Storage size		Size of *RcvDat* in bytes	1 to 4096	Bytes	1
RcvCfg	Reception completion setting		Reception completion setting	---	---	---
Option	Option		Option	---	---	---
Abort	Interruption		Interruption of instruction execution	---	---	FALSE
RcvDat[] (array)	Receive data	In-out	Variable to store data received from the receive buffer	Depends on data type.	---	---
Command-Aborted	Interruption completion	Output	Interruption completion	Depends on data type.	---	---
RcvSize	Receive size		Size of data actually received from the receive buffer	0 to 4096	Bytes	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
DevicePort	Refer to *Function* for details on the structure _sDEVICE_PORT.																			
Size							OK													
RcvCfg	Refer to *Function* for details on the structure _sSERIAL_CFG.																			
Option	Refer to *Function* for details on the structure _sSERIAL_RCV_OPTION.																			
Abort	OK																			
RcvDat[] (array)		OK																		
Command-Aborted	OK																			
RcvSize							OK													

Function

The NX_SerialRcv instruction reads data in No-protocol Mode from the specified port on an NX-series Communications Interface Unit or Option Board.

The data type of the *DevicePort* input variable is structure _sDEVICE_PORT. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
DevicePort	Device port	Object that represents a device port	_sDE-VICE_PORT	---	---	---
DeviceType	Device type	Type of the device to specify	_eDEVICE_-TYPE	_DeviceNXUnit _DeviceEcat-Slave _DeviceOption-Board	---	---
NxUnit	Specified Unit	NX Unit to control	_sNXUNIT_ID	---	---	---
EcatSlave	Specified slave	EtherCAT slave to con-trol	_sECAT_ID	---	---	---
OptBoard	Specified Option Board	Option Board to control	_sOPTBOARD_ID	---	---	---
Reserved	Reserved	Reserved	Reserved	---	---	---
PortNo	Port number	Port number 1: Port 1 2: Port 2	USINT	Depends on data type.	---	---

Use *DeviceType* to specify the device type. Set this to _DeviceNXUnit for an NX Unit and _DeviceOptionBoard for an Option Board. The variable used to specify the device is determined by the specified device type.

To specify an NX Unit, use *NxUnit* to specify the device.

In this case, *EcatSlave* and *OptBoard* are not used.

To *NxUnit*, pass the device variable that is assigned to the node location information on the I/O Map for the device to specify.

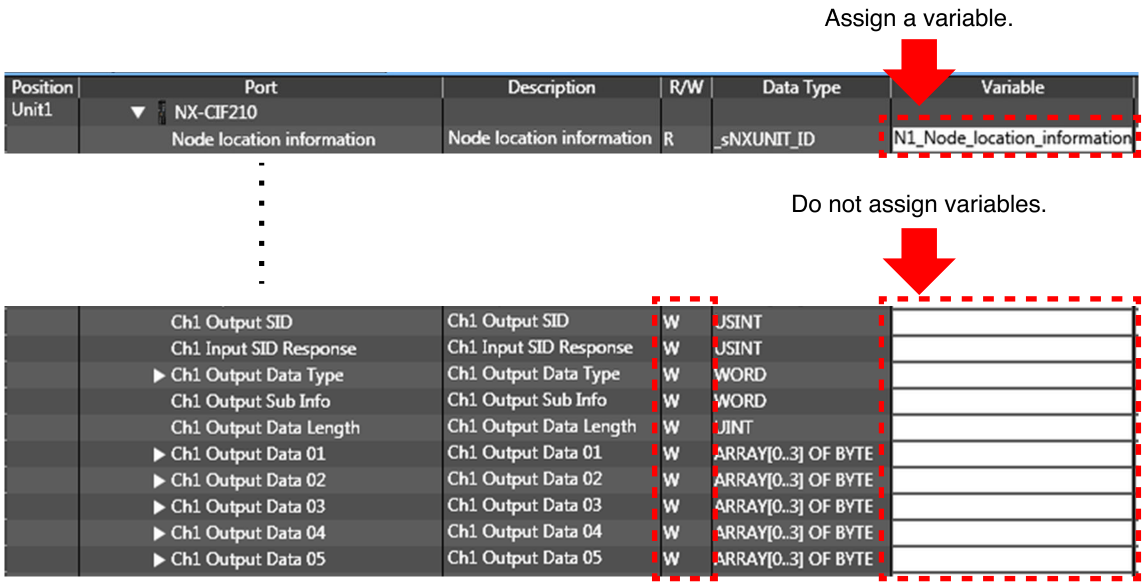
To specify an Option Board, use *OptBoard* to specify the device.

In this case, *NxUnit* and *EcatSlave* are not used.

To *OptBoard*, pass the device variable that is assigned to the node location information on the I/O Map for the device to specify.

If you use this instruction, be sure to assign a device variable to the node location information. Do not assign device variables to any I/O ports following the node location information that are indicated by “W” under the R/W column.

The figure below is an example of using this instruction for port 1 on an NX-CIF210.



Refer to the *Sysmac Studio Version 1 Operation Manual* (Cat. No. W504-E1-07 or later) for details on assigning a device variable to the node location information.

Use *PortNo* to specify the port number.

1: Port 1

2: Port 2

For an NX Unit, set this to Port 1 or Port 2.

For an Option Board, set this to Port 1.

The data type of *DeviceType* is enumerated type _eDEVICE_TYPE.

The meanings of the enumerators of enumerated type _eDEVICE_TYPE are as follows:

Enumerator	Meaning
_DeviceNXUnit	NX Unit is specified.
_DeviceEcatSlave	EtherCAT slave is specified.
_DeviceOptionBoard	Option Board is specified.

In this instruction, you can specify _DeviceNXUnit or _DeviceOptionBoard.

First, data received by the Unit is stored in the receive buffer.

Use the *RcvDat* in-out variable to specify the variable to store data received from the receive buffer.

Use the *Size* input variable to set the size of *RcvDat* in bytes.

The *RcvSize* output variable represents the size of data actually received from the receive buffer.

When the receive data includes start or end code, you must set the *RcvCfg* input variable.
The data type of *RcvCfg* input variable is structure _sSERIAL_CFG. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
RcvCfg	Reception completion setting	Reception completion setting	_sSERIAL_CFG	---	---	---
StartTrig	Start code existence	Start code existence	_eSERIAL_START	_SERIAL_START_NONE _SERIAL_START_STARTCODE1 _SERIAL_START_STARTCODE2	---	_SERIAL_START_NONE
StartCode	Start code	Start code	BYTE[2]	Depends on data type.	---	[2(16#0)]
EndTrig	End code existence	End code existence	_eSERIAL_END	_SERIAL_END_NONE _SERIAL_END_ENDCODE1 _SERIAL_END_ENDCODE2 _SERIAL_END_TERMINATION_CHAR _SERIAL_END_RCV_SIZE	---	_SERIAL_END_NONE
EndCode	End code	End code	BYTE[2]	Depends on data type.	---	[2(16#0)]
RcvSizeCfg	Receive size	Receive size specified when end code is _SERIAL_END_RCV_SIZE	UINT	0 to 4,096	Bytes	0

The data type of *StartTrig* is enumerated type _eSERIAL_START.
The meanings of the enumerators of enumerated type _eSERIAL_START are as follows:

Enumerator	Meaning
_SERIAL_START_NONE	None
_SERIAL_START_STARTCODE1	1-byte code
_SERIAL_START_STARTCODE2	2-byte code

The data type of *EndTrig* is enumerated type _eSERIAL_END.
The meanings of the enumerators of enumerated type _eSERIAL_END are as follows:

Enumerator	Meaning
_SERIAL_END_NONE	None
_SERIAL_END_ENDCODE1	1-byte code
_SERIAL_END_ENDCODE2	2-byte code
_SERIAL_END_TERMINATION_CHAR	Termination condition
_SERIAL_END_RCV_SIZE	Receive size

Refer to *Operation of Start Code and End Code* on page 2-1181 for details on the operation of start code and end code.

To set options, use the *Option* input variable.

The data type of the *Option* input variable is structure _eSERIAL_RCV_OPTION. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Option	Option	Option	_sSERI-AL_RCV_OP-TION	---	---	---
TimeOut*1	Timeout time	Timeout time	UINT	Depends on data type.	0.1 s	20
LastDatRcv (Reserved)	Last data reception	Last data reception	BOOL	FALSE *2	---	FALSE
ClearBuf	Receive buffer clear condition	Receive buffer clear condition	BOOL	Depends on data type.	---	FALSE

*1 An error occurs if the processing does not ends normally within the specified time.
If *TimeOut* is set to 0, the completion of processing will be waited indefinitely.

*2 Always set the value to FALSE.

An error occurs if this instruction is executed for Units other than NX-series Communications Interface Units and Option Boards.

Operation of Start Code and End Code

Use the *RcvCfg.StartTrig* input variable to set the start code condition for the receive data, and use the *RcvCfg.EndTrig* input variable to set the end code condition for the receive data.

The following table shows operation based on combination of *StartTrig* and *EndTrig*.

StartTrig	EndTrig	Operation
_SERIAL_START_NONE	_SERIAL_END_NONE	Data in the receive buffer is received. If there is no receive data in the receive buffer, 0 byte is output to the *RcvSize* output variable and the receive instruction ends normally. If this condition is set, the data of the storage size that is remaining in the receive buffer is read.
	_SERIAL_END_ENDCODE1 _SERIAL_END_ENDCODE2	The following range of data is received from the receive buffer: from the beginning to the end code. Example: ETX
	_SERIAL_END_TERMINATION_CHAR	The following range of data is received from the receive buffer: from the beginning to the data detected as the end. *1
	_SERIAL_END_RCV_SIZE	The following range of data is received from the receive buffer: from the beginning to the receive size specified in *RcvSizeCfg*. Processing is performed only after the specified amount of data is accumulated in the buffer.

StartTrig	EndTrig	Operation
_SERIAL_START_STARTCODE1 _SERIAL_START_STARTCODE2	_SERIAL_END_NONE	The following range of data is received from the receive buffer: from the start code to the end of data.
	_SERIAL_END_ENDCODE1	The following range of data is received from the receive buffer: from the start code to the end code. Example: ETX
	_SERIAL_END_ENDCODE2	
	_SERIAL_END_TERMINA-TION_CHAR	The following range of data is received from the receive buffer: from the start code to the data detected as the end. *1
	_SERIAL_END_RCV_SIZE	The following range of data is received from the receive buffer: from the start code to the receive size specified in *Rcv-Size*. Processing is performed only after the specified amount of data is accumulated in the buffer.

*1 If the number of characters detected as the end of data in the Communications Interface Unit is set to 0 (Do not detect the end), reception will continue until the data of the storage size specified in the *Size* input variable is received.

Precautions for Correct Use

If _SERIAL_END_TERMINATION_CHAR is selected when an Option Board is specified, an error will occur.

Operation When Receive Data Storage Is Insufficient

If the receive data storage specified in the *Size* input variable is smaller than the received data, operation is performed according to the combination of start and end codes, as shown below.

StartTrig	EndTrig	Operation
_SERIAL_START_NONE	_SERIAL_END_NONE	Normal end
	_SERIAL_END_ENDCODE1	Error end, but data is received. Example: ETX
	_SERIAL_END_ENDCODE2	
	_SERIAL_END_TERMINA-TION_CHAR	Error end, but data is received.*1
	_SERIAL_END_RCV_SIZE	• Error end for an input value check error • Data cannot be received.
_SERIAL_START_STARTCODE1 _SERIAL_START_STARTCODE2	_SERIAL_END_NONE	Error end, but data is received.
	_SERIAL_END_ENDCODE1	Error end, but data is received. Example: ETX
	_SERIAL_END_ENDCODE2	
	_SERIAL_END_TERMINA-TION_CHAR	Error end, but data is received.*1
	_SERIAL_END_RCV_SIZE	• Error end for an input value check error • Data cannot be received.

*1 An error occurs if an Option Board is specified.

Data of the size of the storage *RcvDat* is received and the rest of data is retained in the receive buffer. The retained data can be received when the next SerialRcv instruction is executed.

For example, when 10-byte data exists in the receive buffer and the capacity of the receive data storage *RcvDat* is 5 bytes, 5-byte data is received and other 5-byte data is retained in the receive buffer. The value of the *RcvSize* output variable will be 5 bytes, which represents the size of data that is stored.

Receive buffer		Receive data storage RcvDat[ ]
1st byte	Receive processing is performed.	1
2nd byte		2
3rd byte		3
4th byte		4
5th byte		5
6th byte	Cannot be stored in *RcvDat*. Data is retained in the receive buffer. Receive processing for the data is performed when the next NX_SerialRcv instruction is executed.	---
7th byte		
8th byte		
9th byte		
10th byte		

Interruption of Instruction Execution

If *Abort* is changed to TRUE during instruction execution, the execution is interrupted.

When the instruction execution is interrupted, *CommandAborted* changes to TRUE.

If the change of *Abort* is too late to interrupt the execution, *Done* changes to TRUE and the instruction ends normally.

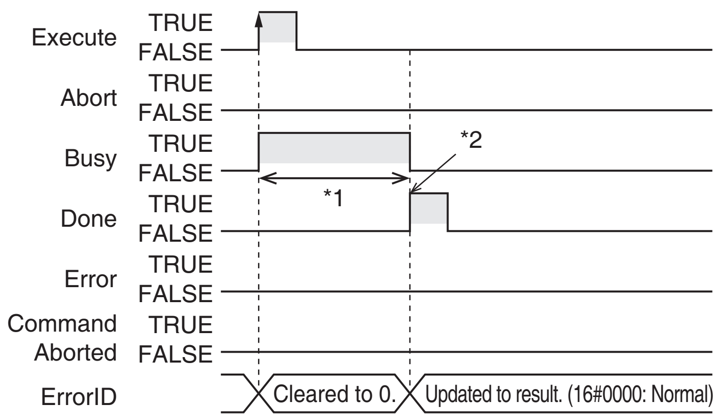
If both *Abort* and *Execute* are changed to TRUE, *CommandAborted* changes to TRUE.

This interruption operation only finishes the *Busy* processing, and it does not clear the receive buffer. To clear the buffer, use the NX_SerialBufClear instruction.

Timing Charts

The following figures show the timing charts.

● Normal end

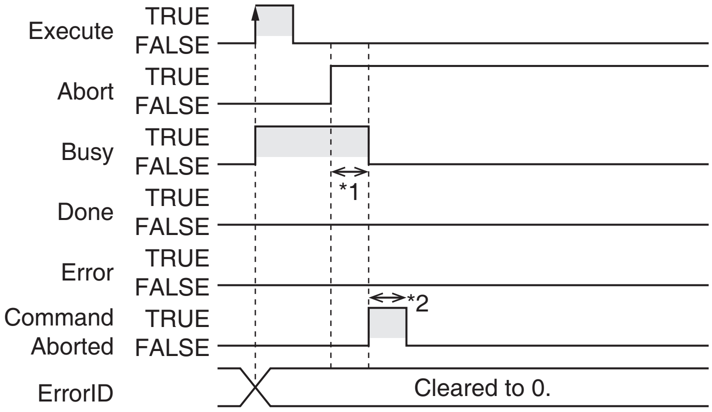


*1 Receive processing

*2 Data is received in No-protocol mode.

● Interruption executed (when *Busy* is TRUE)

The operation is as follows if *Abort* is changed to TRUE while *Busy* is TRUE.

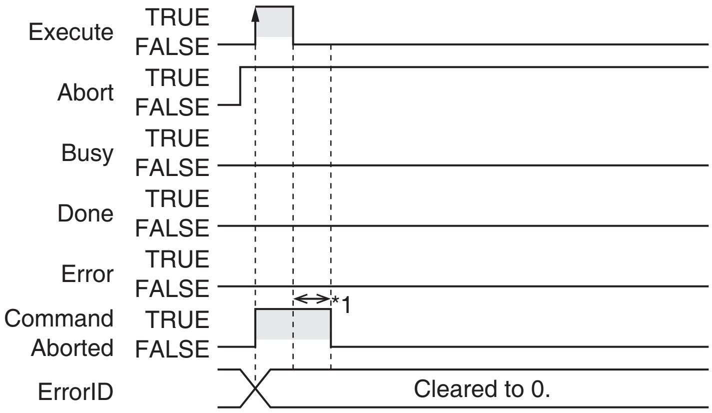


*1 Interruption processing

*2 Changes to FALSE after one task period.

● Interruption executed (when *Execute* is TRUE)

The operation is as follows if both *Abort* and *Execute* are changed to TRUE.



*1 Changes to FALSE after one task period.

Related System-defined Variables

Name	Meaning	Data type	Description
_PLC_OptBoardSta	Option Board Status	ARRAY[1..2] of _sOPTBOARD_STA	• This stores the status of the Option Board.
_NXB_UnitIOActiveTbl	NX Unit I/O Data Active Status	ARRAY[0..8] OF BOOL	• This status tells the NX Units whether I/O data communications can be processed. • The subscript of the array corresponds to the NX Unit numbers. A subscript of 0 means the NX bus master.

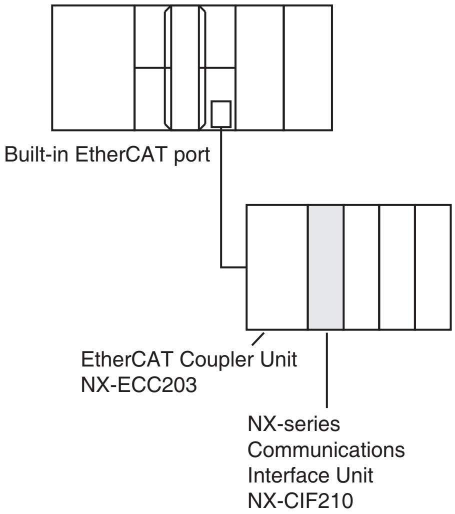
Precautions for Correct Use

- When *Abort* remains FALSE, this instruction is executed until the completion of processing even if *Execute* changes to FALSE or the execution time exceeds the task period.
  The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing. If *Abort* is changed to TRUE during instruction execution, *CommandAborted* or *Done* changes to TRUE.
- Data is not received when *RcvCfg.EndTrig* is _SERIAL_END_RCV_SIZE and the value of the *RcvCfg.RcvSizeCfg* input variable is 0. In this case, the value of *Done* changes to TRUE at instruction execution.
- A compiling error will occur if you use this instruction in an event task. Do not use this instruction in event tasks.
- “CIF Unit Initialized” may occur when the NX-series Communications Interface Unit is restarted. Send or receive the data again, if necessary.
- If you use this instruction, do not assign device variables to any I/O ports that are indicated by “W” under the R/W column on the I/O Map Tab Page in the Sysmac Studio for the applicable NX-series Communications Interface Unit.
- An error will occur in the following cases. *Error* will change to TRUE.
  - A value that is out of range is set for RcvCfg.RcvSizeCfg while *RcvCfg.EndTrig* is set to _SERIAL_END_RCV_SIZE.
  - A value that is out of range is set for *Size*, *DevicePort.DevicePortType* or *DevicePort.PortNo*.
  - *Option.LastDatRcv* is TRUE.
  - The array variable specified with the *RcvDat* in-out variable is smaller than the size specified with the *Size* input variable.
  - The storage size that is specified by *Size* for saving the data in *RcvDat* is smaller than the actually received data.
  - The Unit, Option Board, or port specified with *DevicePort* does not exist.
  - The data type of *DevicePort* is invalid.
  - _SERIAL_END_TERMINATION_CHAR is selected with *RcvCfg.EndTrig* when an Option Board is specified with *DevicePort*.
  - If more than 32 instructions from the NX_SerialSend instruction, NX_SerialRcv instruction, NX_ModbusRtuCmd instruction, NX_ModbusRtuRead instruction, NX_ModbusRtuWrite instruction, NX_SerialSigCtl instruction, NX_SerialSigRead instruction, NX_SerialStatusRead instruction, NX_SerialBufClear instruction, NX_SerialStartMon instruction and NX_SerialStopMon instruction are executed at the same time.
  - The receive buffer is full.
  - This instruction is executed with a device port variable that is the same as the one specified for the instruction which is still being executed. In this case, the instruction which is still being executed is one of the followings. The NX_SerialRcv instruction, NX_ModbusRtuCmd instruction, NX_ModbusRtuRead instruction, and NX_ModbusRtuWrite instruction.
  - A parity error occurred in the data received.
  - A framing error occurred in the data received.
  - An overrun error occurred in the data received.
  - Timeout time elapsed.
  - This instruction is executed for Units other than NX-series Communications Interface Units and Option Boards.
  - The serial communications mode of the specified Option Board is not *No-protocol*.

Sample Programming

In this sample, an NX-series Communications Interface Unit (NX-CIF210) is connected to an EtherCAT Coupler Unit (NX-ECC203).

The unit number of the NX-CIF210 is set to 1.



Data that was read by the barcode reader which is connected to serial port 2 of the NX-CIF210 is obtained.

The receive data is stored in the *RecvDat* in-out variable. There is no start code. End code is 16#OD (CR).

The settings of NX-CIF210 are given in the following table.

Item	Set value
Port 2: Baud Rate	38,400 bps
Port 2: Data Length	8 bits
Port 2: Parity	None
Port 2: Stop Bits	1 bit
Port 2: Flow Control	None

Definitions of Global Variables

Global Variables

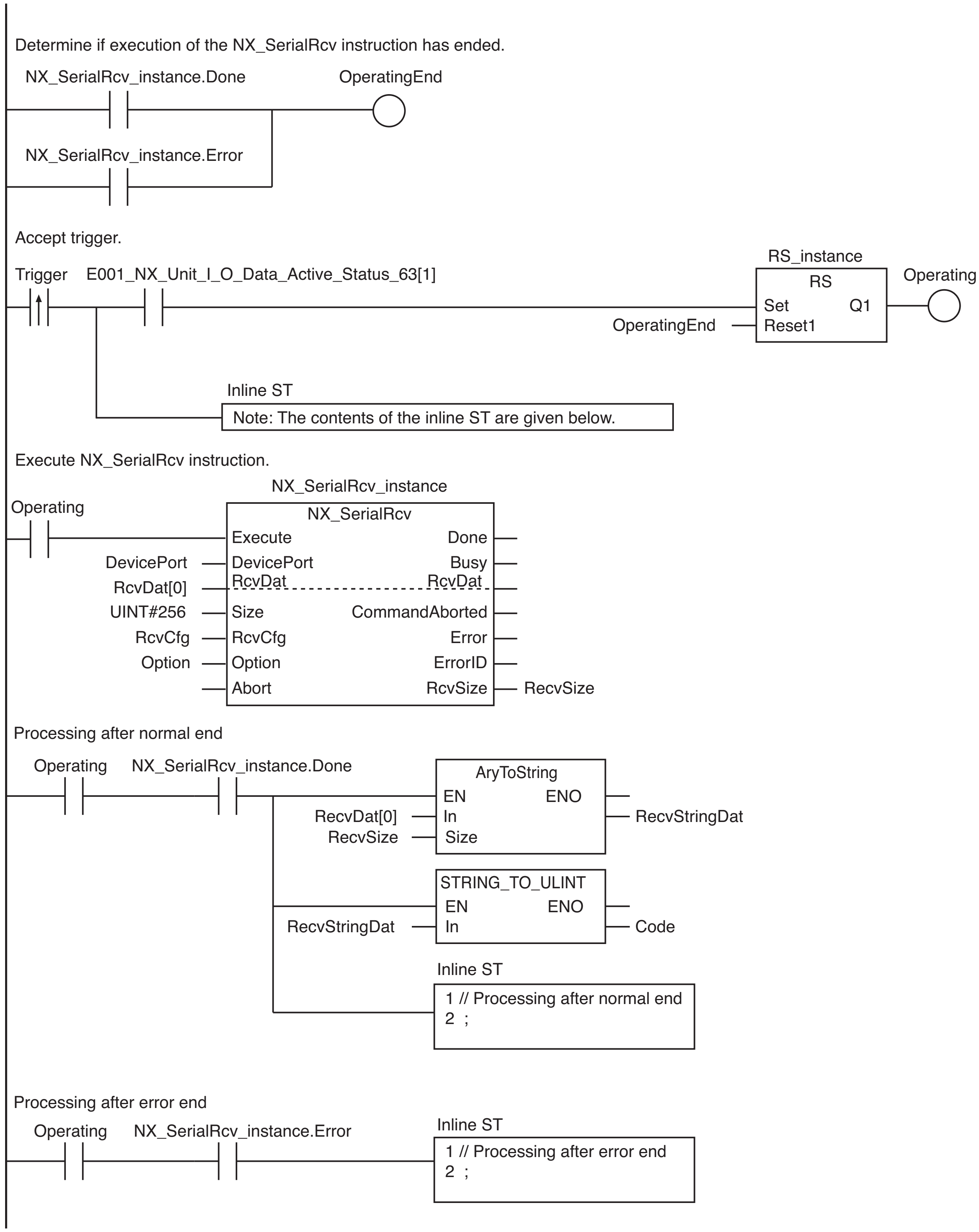
Name	Data type	AT	Comment
E001_NX_Unit_I_O_Data_Ac-tive_Status_63	ARRAY[0..63] OF BOOL	ECAT://node#1/NX Unit I/O Data Active Status 125	Usage of I/O data for 63 NX Units.
N1_Node_location_information	_sNXUNIT_ID	---	Device variable to specify NX-CIF210*1

*1 On the Sysmac Studio, right-click an NX-series slave terminal unit, select **Display Node Location Port**, and set the device variable. Refer to the *Sysmac Studio Version 1 Operation Manual* (Cat. No. W504-E1-07 or later) for details.

LD

Internal Variables	Variable	Data type	Initial value	Comment
	OperationEnd	BOOL	FALSE	Processing completed
	Trigger	BOOL	FALSE	Execution condition
	Operating	BOOL	FALSE	Processing
	DevicePort	_sDEVICE_PORT		Port settings
	RecvDat	ARRAY [0..255] of BYTE	[256(16#0)]	Receive data
	RecvSize	UINT	0	Receive data size
	RecvStringDat	STRING[257]	‘’	
	Code	ULINT	0	Barcode (integer)
	RS_instance	RS		
	NX_SerialRcv_instance	NX_SerialRcv		
	RcvCfg	_sSERIAL_CFG		Reception completion setting
	StartTrig	_eSERIAL_START	_SERIAL_START_NONE	Without start code
	StartCode	BYTE[2]	[2(16#0)]	
	EndTrig	_eSERIAL_END	_SERIAL_END_ENDCODE1	With end code
	EndCode	BYTE[2]	[16#0D,16#00]	16#0D(CR)
	RcvSizeCfg	UINT	0	
	Option	_sSERIAL_RCV_OPTION		Option
	TimeOut	TIME	TIME#0s	
	LastDatRcv	BOOL	FALSE	

External Variables	Variable	Data type	Comment
	E001_NX_Unit_I_O_Data_Active_Status_63	ARRAY[0..63] OF BOOL	• Usage of I/O data for 63 NX Units. • If the relevant Unit number is 1, E001_NX-_Unit_I_O_Data_Active_Status_63[1] is used.
	N1_Node_location_information	_sNXUNIT_ID	Device variable to specify NX-CIF210



Contents of Inline ST

```
DevicePort.DeviceType:=_eDEVICE_TYPE#_DeviceNXUnit;
DevicePort.NxUnit:=N1_Node_location_information;
DevicePort.PortNo:=2;
```

ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	FALSE	Execution condition
	LastTrigger	BOOL	FALSE	Value of Trigger from previous task period
	OperatingStart	BOOL	FALSE	Processing started
	Operating	BOOL	FALSE	Processing
	DevicePort	_sDEVICE_PORT		Port settings
	RecvDat	ARRAY [0..255] of BYTE	[256(16#0)]	Receive data
	RecvSize	UINT	0	Receive data size
	RecvStringDat	STRING[257]	''	
	Code	ULINT	0	Barcode (integer)
	NX_SerialRcv_instance	NX_SerialRcv		
	RcvCfg	_sSERIAL_CFG		Reception completion setting
	StartTrig	_eSERIAL_START	_SERIAL_START_NONE	Without start code
	StartCode	BYTE[2]	[2(16#0)]	
	EndTrig	_eSERIAL_END	_SERIAL_END_ENDCODE1	With end code
	EndCode	BYTE[2]	[16#0D,16#00]	16#0D(CR)
	RcvSizeCfg	UINT	0	
	Option	_sSERIAL_RCV_OPTION		Option
	TimeOut	TIME	TIME#0s	
	LastDatRcv	BOOL	FALSE	

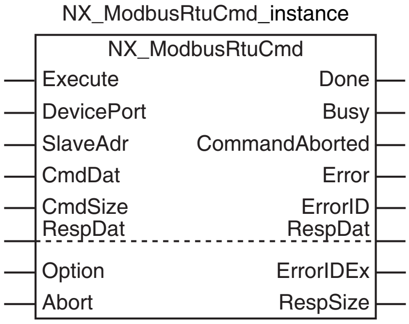
External Variables	Variable	Data type	Comment
	E001_NX_Unit_I_O_Data_Active_Status_63	ARRAY[0..63] OF BOOL	• Usage of I/O data for 63 NX Units. • If the relevant Unit number is 1, E001_NX_Unit_I_O_Data_Active_Status_63[1] is used.
	N1_Node_location_information	_sNXUNIT_ID	Device variable to specify NX-CIF210

```
// Detect when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE)
     AND(E001_NX_Unit_I_O_Data_Active_Status_63[1]) AND (SerialRcv_in-
stance.Busy=FALSE) ) THEN
          OperatingStart:=TRUE;
          Operating:=TRUE;
          DevicePort.DeviceType:=_eDEVICE_TYPE#_DeviceNXUnit;
          DevicePort.NxUnit:=N1_Node_location_information;
          DevicePort.Port.PortNo:=2;
END_IF;
LastTrigger:=Trigger;

// Set communications parameters and initialize SerialRcv instruction.
```

```
IF (OperatingStart=TRUE) THEN
         NX_SerialRcv_instance(
                           Execute:=FALSE,          // Initialize instance.
                           DevicePort:=DevicePort, // Port settings
                           Size:=UINT#256,,         // Receive data size
                           RcvDat:=RecvDat,         // Receive data
                           RcvSize=>RecvSize);      // Data size that was actu-
ally received
         OperatingStart:=FALSE;
END_IF;
// Execute NX_SerialRcv instruction.
IF (Operating=TRUE) THEN
         NX_SerialRcv_instance(
                           Execute:=TRUE,
                           DevicePort:=DevicePort,
                           Size:=UINT#256,
                           RcvDat:=RecvDat,
                           RcvSize=>RecvSize);
         IF (NX_SerialRcv_instance.Done=TRUE) THEN
             // Processing after normal end
             RecvStringDat:=AryToString(In:=RecvDat[0],Size:=RecvSize); // Con-
vert character codes to a text string.
             Code:=STRING_TO_ULINT(RecvDat); // Convert text string to an inte-
ger.

             Operating:=FALSE;
         END_IF;
         IF (NX_SerialRcv_instance.Error=TRUE) THEN
                  // Processing after error end
                  Operating:=FALSE;
         END_IF;
END_IF;
```

NX_ModbusRtuCmd

The NX_ModbusRtuCmd instruction sends general commands from a serial port on an NX-series Communications Interface Unit or Option Board to Modbus-RTU slaves using Modbus-RTU protocol.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
NX_ModbusRtuCmd	Send Modbus RTU General Command	FB		NX_ModbusRtuCmd_instance(Execute, DevicePort, SlaveAdr, CmdDat, CmdSize, RespDat, Option, Abort, Done, Busy, CommandAborted, Error, ErrorID, ErrorIDEx, RespSize);

Version Information

A CPU Unit with unit version 1.11 or later and Sysmac Studio version 1.15 or higher are required to use this instruction.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
DevicePort	Device port	Input	Object that represents a device port	---	---	---
SlaveAdr	Slave address		Address of Modbus-RTU slave*1	0 to 247	---	1
CmdDat[] (array)	Command data		Command data	Depends on data type.	---	*2
CmdSize	Command data size		Command data size	1 to 253	Bytes	*2*3
Option	Option		Option	---	---	---
Abort	Interruption		Interruption of instruction execution	Depends on data type.	---	FALSE
RespDat[] (array)	Read data	In-out	Variable that stores read data	Depends on data type.	---	---
CommandAborted	Interruption completion	Output	Interruption completion	Depends on data type.	---	---
RespSize	Receive size		Receive data size	1 to 253	Bytes	*4

*1 If 0 is set, you can broadcast commands to Modbus-RTU slaves.

*2 If you omit an input parameter, the default value is not applied. A building error will occur.

*3 Set the total number of bytes for the function code and command data. The number of bytes for the function code is one.

*4 The total number of bytes for the function code and read data is stored. The number of bytes for the function code is one.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
DevicePort	Refer to *Function* for details on the structure _sDEVICE_PORT.																			
SlaveAdr							OK													
CmdDat[] (array)		OK																		
CmdSize							OK													
Option	Refer to *Function* for details on the structure _sSERIAL_MODBUSRTU_OPTION.																			
Abort	OK																			
Resp-Dat[]array		OK																		
Command-Aborted	OK																			
RespSize							OK													

Function

The NX_ModbusRtuCmd instruction sends general commands from a serial port on an NX-series Communications Interface Unit or Option Board to Modbus-RTU slaves using Modbus-RTU protocol.

This instruction ends normally when a normal response to the sent command is received.

When a command is broadcasted, this instruction ends normally without waiting for responses from slaves.

The data type of the *DevicePort* input variable is structure _sDEVICE_PORT. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
DevicePort	Device port	Object that represents a device port	_sDEVICE_PORT	---	---	---
DeviceType	Device type	Type of the device to specify	_eDEVICE_TYPE	_DeviceNXUnit _DeviceEcat-Slave _DeviceOption-Board	---	---
NxUnit	Specified Unit	NX Unit to control	_sNXUNIT_ID	---	---	---
EcatSlave	Specified slave	EtherCAT slave to control	_sECAT_ID	---	---	---
OptBoard	Specified Option Board	Option Board to control	_sOPTBOARD_ID	---	---	---
Reserved	Reserved	Reserved	Reserved	---	---	---
PortNo	Port number	Port number 1: Port 1 2: Port 2	USINT	Depends on data type.	---	---

Use *DeviceType* to specify the device type. Set this to _DeviceNXUnit for an NX Unit and _DeviceOptionBoard for an Option Board. The variable used to specify the device is determined by the specified device type.

To specify an NX Unit, use *NxUnit* to specify the device.

In this case, *EcatSlave* and *OptBoard* are not used.

To *NxUnit*, pass the device variable that is assigned to the node location information on the I/O Map for the device to specify.

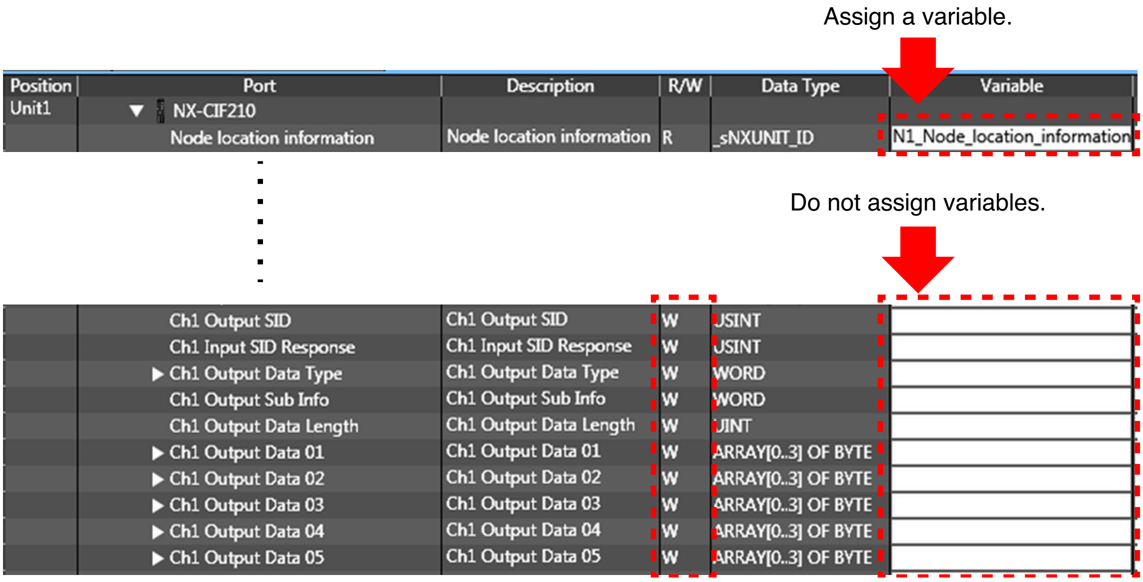
To specify an Option Board, use *OptBoard* to specify the device.

In this case, *NxUnit* and *EcatSlave* are not used.

To *OptBoard*, pass the device variable that is assigned to the node location information on the I/O Map for the device to specify.

If you use this instruction, be sure to assign a device variable to the node location information. Do not assign device variables to any I/O ports following the node location information that are indicated by “W” under the R/W column.

The figure below is an example of using this instruction for port 1 on an NX-CIF210.



Refer to the *Sysmac Studio Version 1 Operation Manual* (Cat. No. W504-E1-07 or later) for details on assigning a device variable to the node location information.

Use *PortNo* to specify the port number.

1: Port 1

2: Port 2

For an NX Unit, set this to Port 1 or Port 2.

For an Option Board, set this to Port 1.

The data type of *DeviceType* is enumerated type _eDEVICE_TYPE.

The meanings of the enumerators of enumerated type _eDEVICE_TYPE are as follows:

Enumerator	Meaning
_DeviceNXUnit	NX Unit is specified.
_DeviceEcatSlave	EtherCAT slave is specified.
_DeviceOptionBoard	Option Board is specified.

In this instruction, you can specify _DeviceNXUnit or _DeviceOptionBoard.

Use the *SlaveAdr* input variable to specify the address of a Modbus-RTU slave.

To broadcast commands to Modbus-RTU slaves, set the *SlaveAdr* input variable to 0.

Set the command data with the *CmdDat* input variable, and set the size of command data with the *CmdSize* input variable.

CRC is attached by the instruction.

Use the *RespDat* in-out variable to specify the variable to store the read data.

The *RespSize* output variable represents the size of received data.

To set options, use the *Option* input variable.

The data type of the *Option* input variable is structure _sSERIAL_MODBUSRTU_OPTION. The specifications are as follows:

Name		Meaning	Description	Data type	Valid range	Unit	Default
Option		Option	Option	_sSERIAL_-MODBUSR-TU_OPTION	---	---	---
	SendDelay	Send delay time	Send delay time in units of 0.01 s	UINT	Depends on data type.	0.01 s	0
	TimeOut	Timeout time	2.0 s when the timeout time is set to 0	UINT	Depends on data type.	0.1 s	20
	NoResponse	No response	• Set TRUE when no response is waited for the send command. • If TRUE is set, this instruction sends a command and ends normally without waiting for the elapse of the timeout time.	BOOL	Depends on data type.	---	FALSE
	Retry	Retry count	Retry count	USINT	0 to 15	---	0

An error occurs if this instruction is executed for Units other than NX-series Communications Interface Units and Option Boards.

Interruption of Instruction Execution

If *Abort* is changed to TRUE during instruction execution, the execution is interrupted.

When the instruction execution is interrupted, *CommandAborted* changes to TRUE.

If the change of *Abort* is too late to interrupt the execution, *Done* changes to TRUE and the instruction ends normally.

If both *Abort* and *Execute* are changed to TRUE, *CommandAborted* changes to TRUE.

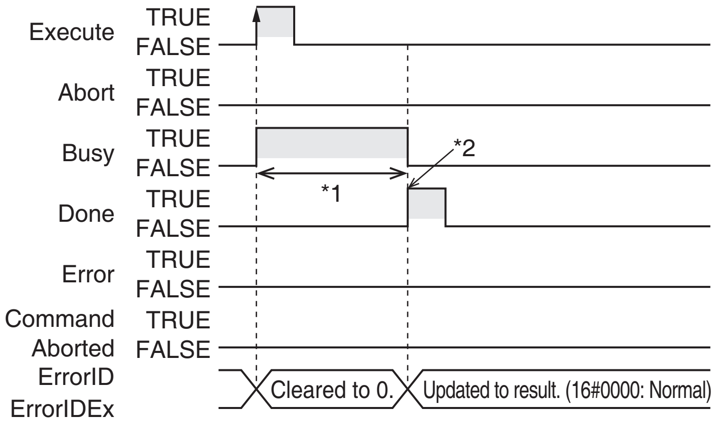
This interruption operation only finishes the *Busy* processing, and it does not clear the send or receive buffer. To clear the buffer, use the NX_SerialBufClear instruction.

Timing Charts

The following figures show the timing charts.

● Normal end (when *SendDelay* is 0 (0 s))

The operation is as follows when *SendDelay* is 0 (0 s).

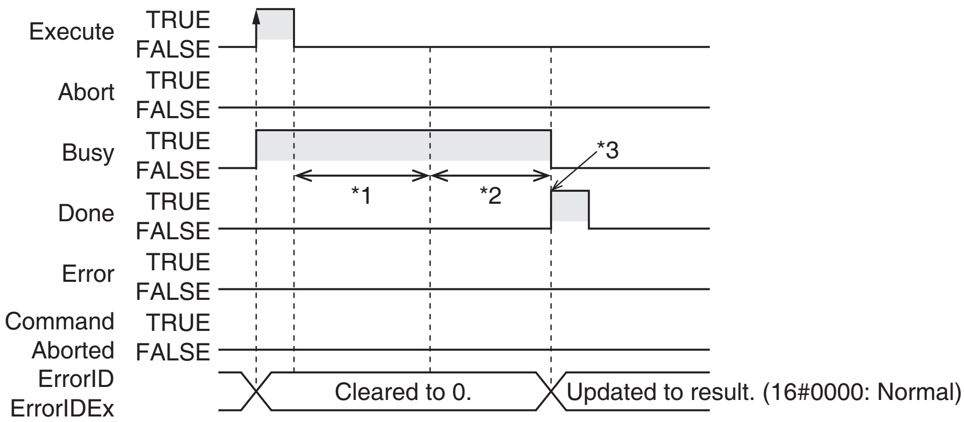


*1 Processing with Modbus-RTU slave

*2 A response to the command is received.

● Normal end (when *SendDelay* is 100 (1 s))

The operation is as follows when *SendDelay* is 100 (1 s).



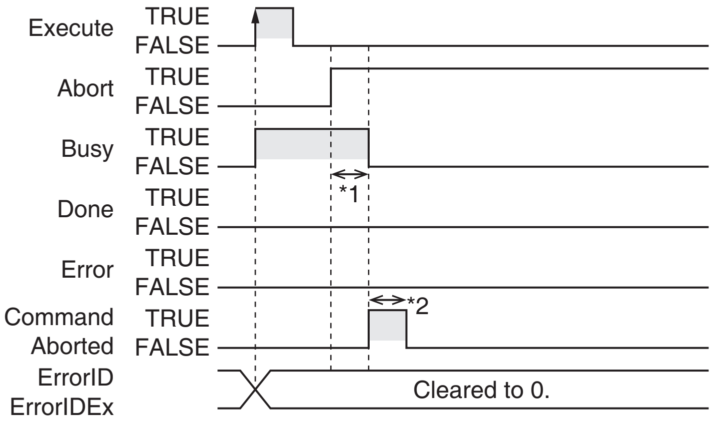
*1 The send delay time of 1 s

*2 A command is sent to a Modbus-RTU slave, and a response is received from the Modbus-RTU slave.

*3 A response to the command is received.

- **Interruption executed (when *Busy* is TRUE)**

The operation is as follows if *Abort* is changed to TRUE while *Busy* is TRUE.

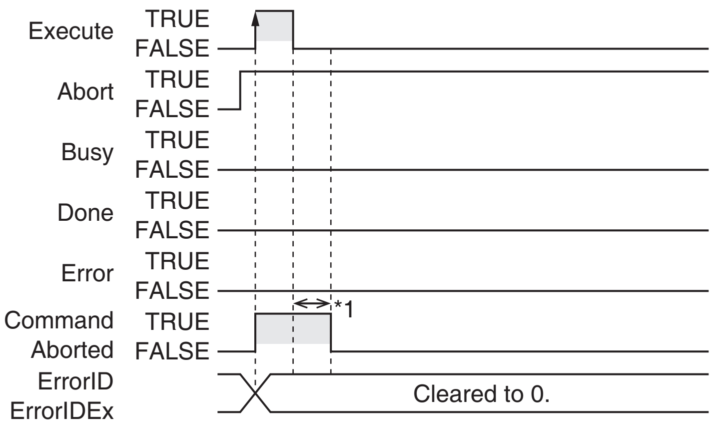


*1 Interruption processing

*2 Changes to FALSE after one task period.

- **Interruption executed (when *Execute* is TRUE)**

The operation is as follows if both *Abort* and *Execute* are changed to TRUE.



*1 Changes to FALSE after one task period.

Related System-defined Variables

Name	Meaning	Data type	Description
_PLC_OptBoardSta	Option Board Status	ARRAY[1..2] of _sOPTBOARD_STA	• This stores the status of the Option Board.
_NXB_UnitIOActiveTbl	NX Unit I/O Data Active Status	ARRAY[0..8] OF BOOL	• This status tells the NX Units whether I/O data communications can be processed. • The subscript of the array corresponds to the NX Unit numbers. A subscript of 0 means the NX bus master.

Additional Information

The frame format used in Modbus-RTU mode is as follows.

Slaves Address	Function Code	Data	CRC
1 byte	1 byte	0 to 252 bytes	2 bytes*

* In CRC code, the low byte comes first, and the high byte comes second.

Refer to the *MODBUS Application Protocol Specification* for the specifications of the MODBUS communications protocol.

You can obtain *MODBUS Application Protocol Specification* from Modbus Organization, Inc.

http://www.modbus.org/

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing. If *Abort* is changed to TRUE during instruction execution, *CommandAborted* or *Done* changes to TRUE.
- A compiling error will occur if you use this instruction in an event task. Do not use this instruction in event tasks.
- "CIF Unit Initialized" may occur when the NX-series Communications Interface Unit is restarted. Send or receive the data again, if necessary.
- If you use this instruction, do not assign device variables to any I/O ports that are indicated by "W" under the R/W column on the I/O Map Tab Page in the Sysmac Studio for the applicable NX-series Communications Interface Unit.
- Data may still remain in the buffer of the target device port in the following cases. To clear the buffer, execute the NX_SerialBufClear instruction before executing the following instruction: NX_ModbusRtuCmd instruction, NX_ModbusRtuRead instruction, or NX_ModbusRtuWrite instruction.
  - After the operation starts or when you change the operating mode to RUN mode.
  - The retry was set (i.e., *Option.Retry* is not 0) in the previous instruction execution.
  - The previous instruction execution is interrupted (i.e., the *CommandAborted* output variable is TRUE).
  - An error occurred (i.e., *Error* is TRUE) in the previous instruction execution.
- An error will occur in the following cases. *Error* will change to TRUE.
  - A value that is out of range was set for *CmdSize*, *Option.Retry*, *DevicePort.DevicePortType*, *DevicePort.PortNo*, or *SlaveAdr*.
  - The variable specified with *CmdDat* is smaller than the size specified with *CmdSize*.
  - The size of the received data is larger than the size of the variable set in *RespDat*.
  - The Unit or port specified with *DevicePort* does not exist.
  - The data type of *DevicePort* is invalid.
  - If more than 32 instructions from the NX_SerialSend instruction, NX_SerialRcv instruction, NX_ModbusRtuCmd instruction, NX_ModbusRtuRead instruction, NX_ModbusRtuWrite instruction, NX_SerialSigCtl instruction, NX_SerialSigRead instruction, NX_SerialStatusRead instruction, NX_SerialBufClear instruction, NX_SerialStartMon instruction and NX_SerialStopMon instruction are executed at the same time.
  - This instruction is executed with a device port variable that is the same as the one specified for the instruction which is still being executed. In this case, the instruction which is still being executed is one of the followings.
    The NX_SerialSend instruction, NX_SerialRcv instruction, NX_ModbusRtuCmd instruction, NX_ModbusRtuRead instruction, and NX_ModbusRtuWrite instruction.
  - A parity error occurred in the data received.

- A framing error occurred in the data received.
- An overrun error occurred in the data received.
- CRC mismatch occurred for the received data.
- Timeout time elapsed.
- This instruction is executed for Units other than NX-series Communications Interface Units and Option Boards.
- An Exception Response was received from a Modbus-RTU slave. You can check Exception Codes with the *ErrorIDEx* output variable.
- There was an invalid function code, receive size, etc. in the response data from a Modbus-RTU slave.
- The serial communications mode of the specified Option Board is not *Modbus-RTU master*.

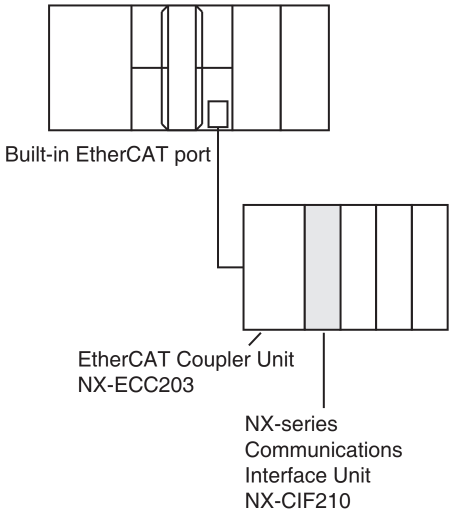
- In this instruction, the expansion error code *ErrorIDEx* is displayed when an error is detected in a Modbus-RTU slave. An expansion error code is output to *ErrorIDEx* when the value of error code *ErrorID* is WORD#16#0C10. The display format is ErrorIDEx=000000XX. For the value XX, refer to the Exception Code specifications of the MODBUS communications protocol.
  Refer to the *MODBUS Application Protocol Specification* for the Exception Code specifications of the MODBUS communications protocol.
  You can obtain *MODBUS Application Protocol Specification* from Modbus Organization, Inc.
  http://www.modbus.org/

Sample Programming

In this sample, an NX-series Communications Interface Unit (NX-CIF210) is connected to an EtherCAT Coupler Unit (NX-ECC203).

The unit number of the NX-CIF210 is set to 1.

For the Unit operation settings of the NX-CIF210, set **Ch2 Number of Characters to Determine the End** to 35. The number of characters is regarded as 3.5 during operation because the unit for setting the Number of Characters to Determine the End is 0.1 character.



When *Trigger* changes to TRUE, the instruction clears the buffer of the serial port 2 on the NX-CIF210 and then sends a Modbus-RTU command.

It reads a holding register from the read start address 32 (BYTE#16#0020) in slave address 1.

General commands are sent/received to read a variable.

Internal Variables	Variable	Data type	Initial value	Comment
	Stage	INT	0	
	Trigger	BOOL	FALSE	Execution condition
	DevicePort	_sDEVICE_PORT		Port settings
	NX_SerialBufClear_instance	NX_SerialBufClear		Clear buffer
	ClearDone	BOOL		
	ClearError	BOOL		
	NX_ModbusRtuCmd_instance	NX_ModbusRtuCmd		
	ModbusSlaveAdr	UINT	UINT#0	Slave address
	ModbusCmdDat	ARRAY[0..19] OF BYTE		Modbus command data
	ModbusDatSize	UINT	UINT#0	Modbus command data total size (byte)
	ModbusRespDat	ARRAY[0..275] OF BYTE		Received data storage area
	ModbusDone	BOOL		
	ModbusCommandAborted	BOOL		

Internal Variables	Variable	Data type	Initial value	Comment
	ModbusError	BOOL		
	ModbusRspSize	UINT		Actually received data size (byte)
	DoModbusTrigger	BOOL		

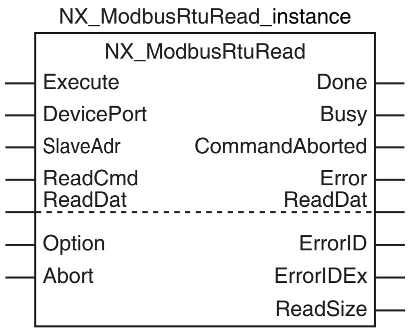
External Variables	Variable	Data type	Constant	Comment
	N1_Node_location_information	_sNXUNIT_ID	✔	

```
// Start sequence when Trigger changes to TRUE.
IF (Trigger=TRUE) AND (DoModbusTrigger=FALSE) THEN
    DoModbusTrigger := TRUE;

    NX_SerialBufClear_instance(Execute := FALSE,
        DevicePort:=DevicePort );
    NX_ModbusRtuCmd_instance(Execute:= FALSE,
        DevicePort:=DevicePort,
        CmdDat:=ModbusCmdDat[1],
        CmdSize:=ModbusDatSize,
        RespDat:=ModbusRespDat[0] );
    Stage := 1;        // Initialization completed.
END_IF;

IF (DoModbusTrigger=TRUE) THEN
    CASE Stage OF
    1:    // Buffer clear request
        DevicePort.DeviceType:=_eDEVICE_TYPE#_DeviceNXUnit;
        DevicePort.NxUnit:=N1_Node_location_information;
        DevicePort.PortNo:=2;

        NX_SerialBufClear_instance(Execute := TRUE,
            DevicePort:=DevicePort,
            Done => ClearDone,
            Error => ClearError);

        IF (ClearDone = TRUE) THEN
            Stage := 2;        // Buffer clear is normal end.
        ELSIF ( ClearError = TRUE ) THEN
            Stage := 99;        // Buffer clear is error end.
        END_IF;

    2:    // Modbus  Cmd send request
        ModbusSlaveAdr := 1;                // Slave address
        ModbusCmdDat[1]:=BYTE#16#03;        // Function code (read variable)
        ModbusCmdDat[2]:=BYTE#16#00;        // Read start address (H)
        ModbusCmdDat[3]:=BYTE#16#20;        // Read start address (L)
        ModbusCmdDat[4]:=BYTE#16#00;        // Number of data (H)
        ModbusCmdDat[5]:=BYTE#16#01;        // Number of data (L)
        ModbusDatSize:=5;

        NX_ModbusRtuCmd_instance(Execute:= TRUE,
            DevicePort:=DevicePort,
            SlaveAdr:=ModbusSlaveAdr,
            CmdDat:=ModbusCmdDat[1],
```

```
            CmdSize:=ModbusDatSize,
            RespDat:=ModbusRespDat[0],
            Done=>ModbusDone,
            CommandAborted=>ModbusCommandAborted,
            Error=>ModbusError,
            RespSize=>ModbusRspSize);

        IF (ModbusDone = TRUE) THEN
            Stage := 3;        // The NX_ModbusRtuCmd instruction is normal end.
        ELSIF (ModbusError=TRUE) OR (ModbusCommandAborted=TRUE) THEN
             Stage :=99;        // The NX_ModbusRtuCmd instruction is error end or
Abort.
        END_IF;

    3:    // Processing after the NX_ModbusRtuCmd instruction is normal end.
        Trigger := FALSE;
        DoModbusTrigger := FALSE;

    99:    // Error Processing
        Trigger := FALSE;
        DoModbusTrigger := FALSE;
    END_CASE;
END_IF;
```

NX_ModbusRtuRead

The NX_ModbusRtuRead instruction sends read commands from a serial port on an NX-series Communications Interface Unit or Option Board to Modbus-RTU slaves using Modbus-RTU protocol.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
NX_ModbusR-tuRead	Send Modbus RTU Read Command	FB		NX_ModbusRtuRead_instance(Execute, DevicePort, SlaveAdr, ReadCmd, ReadDat, Option, Abort, Done, Busy, CommandAborted, Error, ErrorID, ErrorIDEx, ReadSize);

Version Information

A CPU Unit with unit version 1.11 or later and Sysmac Studio version 1.15 or higher are required to use this instruction.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
DevicePort	Device port	Input	Object that represents a device port	---	---	---
SlaveAdr	Slave address		Address of Modbus-RTU slave*1	1 to 247	---	1
ReadCmd	Read command		Read command	---	---	*2
Option	Option		Option	---	---	---
Abort	Interruption		Interruption of instruction execution	Depends on data type.	---	FALSE
ReadDat[] (array)	Read data	In-out	Variable that stores read data	Depends on data type.	---	---
Command-Aborted	Interruption completion	Output	Interruption completion	Depends on data type.	---	---
ReadSize	Receive size		Receive data size	1 to 2,000*3	---*4	---

*1 An error occurs if 0 is set.

*2 If you omit an input parameter, the default value is not applied. A building error will occur.

*3 If receive data is WORD data, the upper limit value is 125.

*4 The unit is the same as the unit of read data specified with *ReadCmd.Fun.*

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
DevicePort	Refer to *Function* for details on the structure _sDEVICE_PORT.																			
SlaveAdr							OK													
ReadCmd	Refer to *Function* for details on the structure _sSERIAL_MODBUSRTU_READ.																			
Option	Refer to *Function* for details on the structure _sSERIAL_MODBUSRTU_OPTION.																			
Abort	OK																			
ReadDat[] (array)	OK		OK																	
	An array can also be specified.																			
Command-Aborted	OK																			
ReadSize							OK													

Function

The NX_ModbusRtuRead instruction sends read commands from a serial port on an NX-series Communications Interface Unit or Option Board to Modbus-RTU slaves using Modbus-RTU protocol. The requested data are read from the Modbus-RTU slaves.

This instruction ends normally when a normal response to the sent command is received.

The data type of the *DevicePort* input variable is structure _sDEVICE_PORT. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
DevicePort	Device port	Object that represents a device port	_sDEVICE_PORT	---	---	---
DeviceType	Device type	Type of the device to specify	_eDEVICE_-TYPE	_DeviceNXUnit _DeviceEcat-Slave _DeviceOption-Board	---	---
NxUnit	Specified Unit	NX Unit to control	_sNXUNIT_ID	---	---	---
EcatSlave	Specified slave	EtherCAT slave to control	_sECAT_ID	---	---	---
OptBoard	Specified Option Board	Option Board to control	_sOPTBOARD_ID	---	---	---
Reserved	Reserved	Reserved	Reserved	---	---	---
PortNo	Port number	Port number 1: Port 1 2: Port 2	USINT	Depends on data type.	---	---

Use *DeviceType* to specify the device type. Set this to _DeviceNXUnit for an NX Unit and _DeviceOptionBoard for an Option Board. The variable used to specify the device is determined by the specified device type.

To specify an NX Unit, use *NxUnit* to specify the device.

In this case, *EcatSlave* and *OptBoard* are not used.

To *NxUnit*, pass the device variable that is assigned to the node location information on the I/O Map for the device to specify.

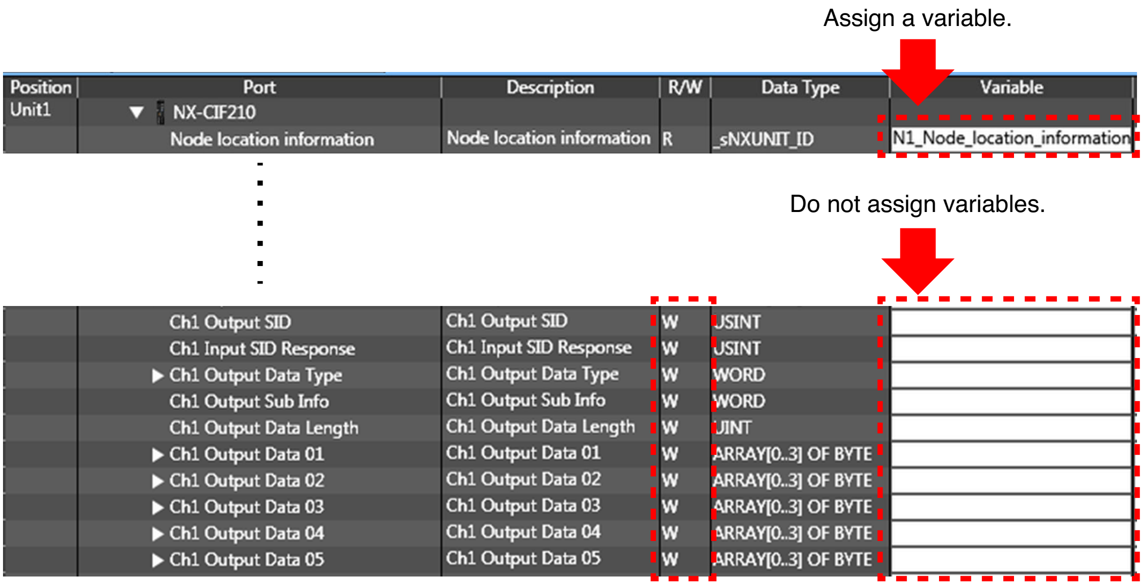
To specify an Option Board, use *OptBoard* to specify the device.

In this case, *NxUnit* and *EcatSlave* are not used.

To *OptBoard*, pass the device variable that is assigned to the node location information on the I/O Map for the device to specify.

If you use this instruction, be sure to assign a device variable to the node location information. Do not assign device variables to any I/O ports following the node location information that are indicated by "W" under the R/W column.

The figure below is an example of using this instruction for port 1 on an NX-CIF210.



Refer to the *Sysmac Studio Version 1 Operation Manual* (Cat. No. W504-E1-07 or later) for details on assigning a device variable to the node location information.

Use *PortNo* to specify the port number.

1: Port 1

2: Port 2

For an NX Unit, set this to Port 1 or Port 2.

For an Option Board, set this to Port 1.

The data type of *DeviceType* is enumerated type _eDEVICE_TYPE.

The meanings of the enumerators of enumerated type _eDEVICE_TYPE are as follows:

Enumerator	Meaning
_DeviceNXUnit	NX Unit is specified.
_DeviceEcatSlave	EtherCAT slave is specified.
_DeviceOptionBoard	Option Board is specified.

In this instruction, you can specify _DeviceNXUnit or _DeviceOptionBoard.

Use the *SlaveAdr* input variable to specify the address of a Modbus-RTU slave.

If 0 is set for the *SlaveAdr* input variable, an error occurs and you cannot broadcast commands to Modbus-RTU slaves.

Use the *ReadCmd* input variable to specify the read command.

CRC is attached by the instruction.

The data type of *ReadCmd* input variable is structure _sSERIAL_MODBUSRTU_READ. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
ReadCmd	Read command	Read command	_sSERIAL_MODBUSRTU_READ	---	---	---
Fun	Function code	Function code	_eMDB_FUN	_MDB_READ_COILS _MDB_READ_DISCRETE_INPUTS _MDB_READ_HOLDING_REGISTERS _MDB_READ_INPUT_REGISTERS	---	_MDB_READ_COILS
ReadAdr	Read address	Read start address	UINT	Depends on data type.	---	0
ReadSize	Read size	Read size	UINT	Depends on function code.	---*1	_MDB_READ_COILS

*1 The unit is the same as the unit of read data specified with *ReadCmd.Fun.*

The data type of *Fun* is enumerated type _eMDB_FUN.

The meanings of the enumerators of enumerated type _eMDB_FUN are as follows:

Enumerator	Meaning
_MDB_READ_COILS	Read outputs (bit)
_MDB_READ_DISCRETE_INPUTS	Read inputs (bit)
_MDB_READ_HOLDING_REGISTERS	Read holding registers (word)
_MDB_READ_INPUT_REGISTERS	Read input registers (word)

The valid range that you can specify with *ReadSize* varies depending on the function code.

Each value is determined by the size of data that is read and the maximum command length.

The specifications are as follows:

Function code	ReadSize
_MDB_READ_COILS	1 to 2,000 (bit)
_MDB_READ_DISCRETE_INPUTS	1 to 2,000 (bit)
_MDB_READ_HOLDING_REGISTERS	1 to 125 (word)
_MDB_READ_INPUT_REGISTERS	1 to 125 (word)

Use the *ReadDat* in-out variable to specify the variable to store the read data.

The data type that you can use for *ReadDat* differs depending on the function code.

The specifications are as follows:

Function code	Data type
_MDB_READ_COILS	BOOL BOOL[]
_MDB_READ_DISCRETE_INPUTS	BOOL BOOL[]
_MDB_READ_HOLDING_REGISTERS	WORD WORD[]
_MDB_READ_INPUT_REGISTERS	WORD WORD[]

The *ReadSize* output variable represents the size of data that was read.

To set options, use the *Option* input variable.

The data type of the *Option* input variable is structure _sSERIAL_MODBUSRTU_OPTION. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Option	Option	Option	_sSERIAL_ MODBUSR-TU_ OPTION	---	---	---
SendDelay	Send delay time	Send delay time	UINT	Depends on data type.	0.01 s	0
TimeOut	Timeout time	2.0 s when the timeout time is set to 0	UINT	Depends on data type.	0.1 s	20
NoResponse	No response	Not used in this instruction.	BOOL	Depends on data type.	---	FALSE
Retry	Retry count	Retry count	USINT	0 to 15	---	0

An error occurs if this instruction is executed for Units other than NX-series Communications Interface Units and Option Boards.

Interruption of Instruction Execution

If *Abort* is changed to TRUE during instruction execution, the execution is interrupted.

When the instruction execution is interrupted, *CommandAborted* changes to TRUE.

If the change of *Abort* is too late to interrupt the execution, *Done* changes to TRUE and the instruction ends normally.

If both *Abort* and *Execute* are changed to TRUE, *CommandAborted* changes to TRUE.

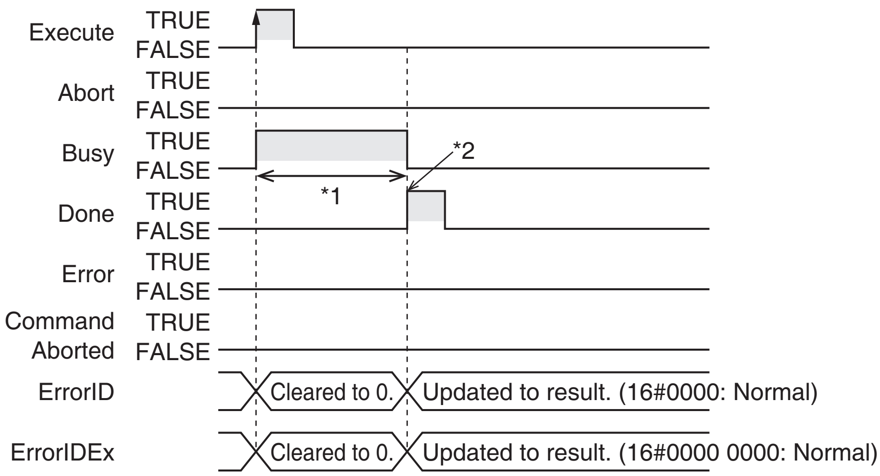
This interruption operation only finishes the *Busy* processing for the instruction, and it does not clear the send or receive buffer. To clear the buffer, use the NX_SerialBufClear instruction.

Timing Charts

The following figures show the timing charts.

● Normal end (when *SendDelay* is 0 (0 s))

The operation is as follows when *SendDelay* is 0 (0 s).

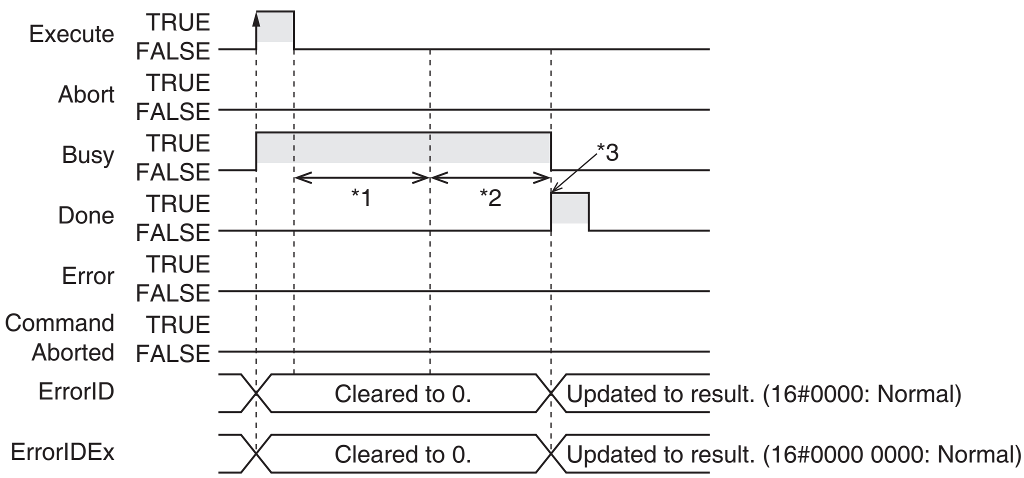


*1 Processing with Modbus-RTU slave

*2 A response to the command is received.

● Normal end (when *SendDelay* is 100 (1 s))

The operation is as follows when *SendDelay* is 100 (1 s).



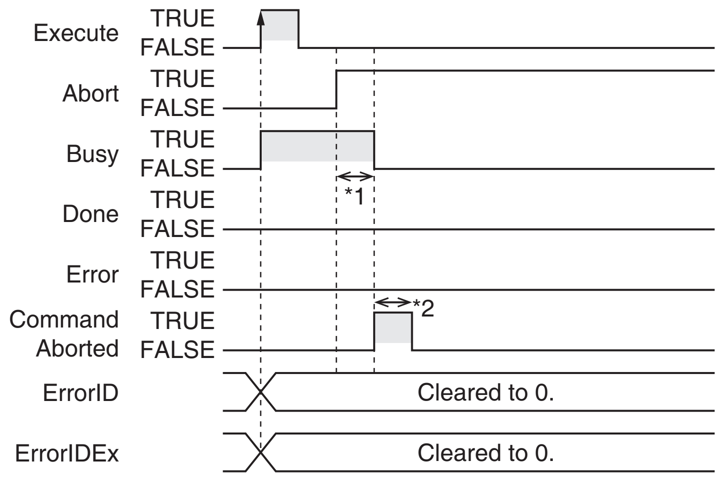
*1 The send delay time of 1 s

*2 A read command is sent to Modbus-RTU slave, and a response is received from Modbus-RTU slave.

*3 A response to the command is received.

● **Interruption executed (when *Busy* is TRUE)**

The operation is as follows if *Abort* is changed to TRUE while *Busy* is TRUE.

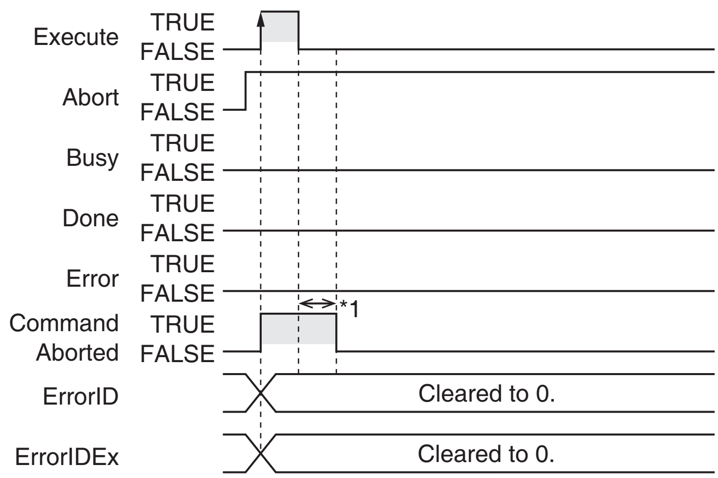


*1 Interruption processing

*2 Changes to FALSE after one task period.

● **Interruption executed (when *Execute* is TRUE)**

The operation is as follows if both *Abort* and *Execute* are changed to TRUE.



*1 Changes to FALSE after one task period.

Related System-defined Variables

Name	Meaning	Data type	Description
_PLC_OptBoardSta	Option Board Status	ARRAY[1..2] of _sOPTBOARD_STA	• This stores the status of the Option Board.
_NXB_UnitIOActiveTbl	NX Unit I/O Data Active Status	ARRAY[0..8] OF BOOL	• This status tells the NX Units whether I/O data communications can be processed. • The subscript of the array corresponds to the NX Unit numbers. A subscript of 0 means the NX bus master.

Additional Information

Refer to the *MODBUS Application Protocol Specification* for the specifications of the MODBUS communications protocol.

You can obtain *MODBUS Application Protocol Specification* from Modbus Organization, Inc.

http://www.modbus.org/

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing. If *Abort* is changed to TRUE during instruction execution, *CommandAborted* or *Done* changes to TRUE.
- A compiling error will occur if you use this instruction in an event task. Do not use this instruction in event tasks.
- “CIF Unit Initialized” may occur when the NX-series Communications Interface Unit is restarted. Send or receive the data again, if necessary.
- If you use this instruction, do not assign device variables to any I/O ports that are indicated by “W” under the R/W column on the I/O Map Tab Page in the Sysmac Studio for the applicable NX-series Communications Interface Unit.
- Data may still remain in the buffer of the target device port in the following cases. To clear the buffer, execute the NX_SerialBufClear instruction before executing the following instruction: NX_ModbusRtuCmd instruction, NX_ModbusRtuRead instruction, or NX_ModbusRtuWrite instruction.
  - After the operation starts or when you change the operating mode to RUN mode.
  - The retry was set (i.e., *Option.Retry* is not 0) in the previous instruction execution.
  - The previous instruction execution is interrupted (i.e., the *CommandAborted* output variable is TRUE).
  - An error occurred (i.e., *Error* is TRUE) in the previous instruction execution.
- An error will occur in the following cases. *Error* will change to TRUE.
  - A value that is out of range was set for *SlaveAdr*, *ReadCmd.ReadSize*, *ReadCmd.Fun*, *Option.Retry*, *DevicePort.DevicePortType*, or *DevicePort.PortNo*.
  - The variable specified with *ReadDat* is smaller than the size specified with *ReadCmd.ReadSize*.
  - The Unit or port specified with *DevicePort* does not exist.
  - The data type of *DevicePort* or *RespDat* is invalid.
  - If more than 32 instructions from the NX_SerialSend instruction, NX_SerialRcv instruction, NX_ModbusRtuCmd instruction, NX_ModbusRtuRead instruction, NX_ModbusRtuWrite instruction, NX_SerialSigCtl instruction, NX_SerialSigRead instruction, NX_SerialStatusRead instruction, NX_SerialBufClear instruction, NX_SerialStartMon instruction and NX_SerialStopMon instruction are executed at the same time.
  - This instruction is executed with a device port variable that is the same as the one specified for the instruction which is still being executed. In this case, the instruction which is still being executed is one of the followings.
    The NX_SerialSend instruction, NX_SerialRcv instruction, NX_ModbusRtuCmd instruction, NX_ModbusRtuRead instruction, and NX_ModbusRtuWrite instruction.
  - A parity error occurred in the data received.
  - A framing error occurred in the data received.
  - An overrun error occurred in the data received.
  - CRC mismatch occurred for the received data.
  - Timeout time elapsed. (When the retry is set, timeout time is multiplied by the number of retries.)
  - This instruction is executed for Units other than NX-series Communications Interface Units and Option Boards.

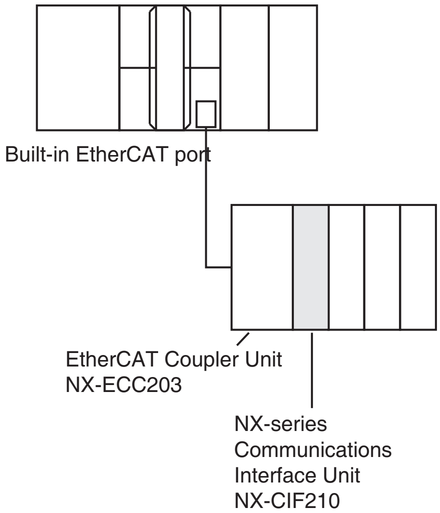
   - An Exception Response was received from a Modbus-RTU slave. You can check Exception Codes with the *ErrorIDEx* output variable.
   - There was an invalid function code, receive size, etc. in the response data from a Modbus-RTU slave.
   - The serial communications mode of the specified Option Board is not *Modbus-RTU master*.
- In this instruction, the expansion error code *ErrorIDEx* is displayed when an error is detected in a Modbus-RTU slave. An expansion error code is output to *ErrorIDEx* when the value of error code *ErrorID* is WORD#16#0C10. The display format is ErrorIDEx=000000XX. For the value XX, refer to the Exception Code specifications of the MODBUS communications protocol.
  Refer to the *MODBUS Application Protocol Specification* for the Exception Code specifications of the MODBUS communications protocol.
  You can obtain *MODBUS Application Protocol Specification* from Modbus Organization, Inc.
  http://www.modbus.org/

Sample Programming

In this sample, an NX-series Communications Interface Unit (NX-CIF210) is connected to an EtherCAT Coupler Unit (NX-ECC203).

The unit number of the NX-CIF210 is set to 1.

For the Unit operation settings of the NX-CIF210, set **Ch2 Number of Characters to Determine the End** to 35. The number of characters is regarded as 3.5 during operation because the unit for setting the Number of Characters to Determine the End is 0.1 character.



When *Trigger* changes to TRUE, the instruction clears the buffer of the serial port 2 on the NX-CIF210 and then sends a Modbus-RTU command.

It reads the status of an output from the read start address 19 in slave address 1.

A read command is sent to read a variable.

Internal Variables	Variable	Data type	Initial value	Comment
	Stage	INT	0	
	Trigger	BOOL	FALSE	Execution condition
	DevicePort	_sDEVICE_PORT		Port settings
	NX_SerialBufClear_instance	NX_SerialBufClear		Clear buffer
	ClearDone	BOOL		
	ClearError	BOOL		
	NX_ModbusRtuRead_instance	NX_ModbusRtuRead		
	ModbusSlaveAdr	UINT	UINT#0	Slave address
	ModbusDone	BOOL		
	ModbusCommandAborted	BOOL		
	ModbusError	BOOL		
	ModbusReadSize	UINT		Actually received data size (byte)

Internal Variables	Variable	Data type	Initial value	Comment
	DoModbusTrigger	BOOL		
	ModbusReadDat	BOOL		
	ModbusReadCmd	_sSERIAL_MODBUSRTU_READ		

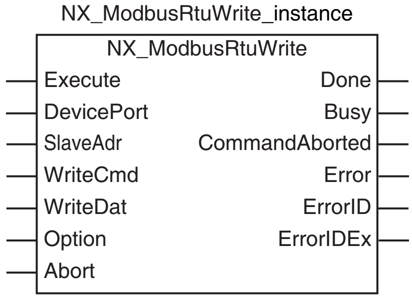
External Variables	Variable	Data type	Constant	Comment
	N1_Node_location_information	_sNXUNIT_ID	✔	

```
// Start sequence when Trigger changes to TRUE.
IF (Trigger=TRUE) AND (DoModbusTrigger=FALSE) THEN
    DoModbusTrigger := TRUE;

    NX_SerialBufClear_instance(Execute := FALSE,
        DevicePort:=DevicePort);
    NX_ModbusRtuRead_instance(Execute:= FALSE,
        DevicePort:=DevicePort,
        ReadDat:=ModbusReadDat);
    Stage := 1;        // Initialization completed.
END_IF;

IF (DoModbusTrigger=TRUE) THEN
    CASE Stage OF
    1:    // Buffer clear request
        DevicePort.DeviceType:=_eDEVICE_TYPE#_DeviceNXUnit;
        DevicePort.NxUnit:=N1_Node_location_information;
        DevicePort.PortNo:=2;

        NX_SerialBufClear_instance(Execute := TRUE,
            DevicePort:=DevicePort,
            Done => ClearDone,
            Error => ClearError);

        IF (ClearDone = TRUE) THEN
            Stage := 2;        // Buffer clear is normal end.
        ELSIF (ClearError = TRUE) THEN
            Stage := 99;        // Buffer clear is error end.
        END_IF;

    2:    // Modbus read request
        ModbusSlaveAdr := 1;                    // Slave address
        ModbusReadCmd.Fun:=_MDB_READ_COILS;     // Function code
        ModbusReadCmd.ReadAdr:=19;              // Read address
        ModbusReadCmd.ReadSize:=1;              // Read size

        NX_ModbusRtuRead_instance(Execute:= TRUE,
            DevicePort:=DevicePort,
            SlaveAdr:=ModbusSlaveAdr,
            ReadCmd:=ModbusReadCmd,
            ReadDat:=ModbusReadDat,
            Done=>ModbusDone,
            CommandAborted=>ModbusCommandAborted,
            Error=>ModbusError,
            ReadSize=>ModbusReadSize);

        IF (ModbusDone = TRUE) THEN
```

```
            Stage := 3;        // The NX_ModbusRead instruction is normal end.
        ELSIF (ModbusError=TRUE) OR (ModbusCommandAborted=TRUE) THEN
             Stage :=99;         // The NX_ModbusRead instruction is error end or
Abort.
        END_IF;

    3:    // Processing after the NX_ModbusRead instruction is normal end.
        Trigger := FALSE;
        DoModbusTrigger := FALSE;

    99:    // Error Processing
        Trigger := FALSE;
        DoModbusTrigger := FALSE;
    END_CASE;
END_IF;
```

NX_ModbusRtuWrite

The NX_ModbusRtuWrite instruction sends write commands from a serial port on an NX-series Communications Interface Unit or Option Board to Modbus-RTU slaves using Modbus-RTU protocol.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
NX_ModbusRtuWrite	Send Modbus RTU Write Command	FB		NX_ModbusRtuWrite_instance(Execute, DevicePort, SlaveAdr, WriteCmd, WriteDat, Option, Abort, Done, Busy, CommandAborted, Error, ErrorID, ErrorIDEx);

Version Information

A CPU Unit with unit version 1.11 or later and Sysmac Studio version 1.15 or higher are required to use this instruction.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
DevicePort	Device port	Input	Object that represents a device port	---	---	---
SlaveAdr	Slave address		Address of Modbus-RTU slave*1	0 to 247	---	1
WriteCmd	Write command		Write command	---	---	*2
WriteDat[] (array)	Write data		Write data	Depends on data type.	---	*2
Option	Option		Option	---	---	---
Abort	Interruption		Interruption of instruction execution	Depends on data type.	---	FALSE
CommandAborted	Interruption completion	Output	Interruption completion	Depends on data type.	---	---

*1 If 0 is set, you can broadcast commands to Modbus-RTU slaves.

*2 If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
DevicePort	Refer to *Function* for details on the structure _sDEVICE_PORT.																			
SlaveAdr							OK													
WriteCmd	Refer to *Function* for details on the structure _sSERIAL_MODBUSRTU_WRITE.																			
WriteDat[] (array)	OK		OK																	
	An array can also be specified.																			
Option	Refer to *Function* for details on the structure _sSERIAL_MODBUSRTU_OPTION.																			
Abort	OK																			
Command-Aborted	OK																			

Function

The NX_ModbusRtuWrite instruction sends write commands from a serial port on an NX-series Communications Interface Unit or Option Board to Modbus-RTU slaves using Modbus-RTU protocol.

This instruction ends normally when a normal response to the sent command is received.

When a command is broadcasted, this instruction ends normally without waiting for responses from slaves.

The data type of the *DevicePort* input variable is structure _sDEVICE_PORT. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
DevicePort	Device port	Object that represents a device port	_sDEVICE_PORT	---	---	---
DeviceType	Device type	Type of the device to specify	_eDEVICE_TYPE	_DeviceNXUnit _DeviceEcatSlave _DeviceOptionBoard	---	---
NxUnit	Specified Unit	NX Unit to control	_sNXUNIT_ID	---	---	---
EcatSlave	Specified slave	EtherCAT slave to control	_sECAT_ID	---	---	---
OptBoard	Specified Option Board	Option Board to control	_sOPTBOARD_ID	---	---	---
Reserved	Reserved	Reserved	Reserved	---	---	---
PortNo	Port number	Port number 1: Port 1 2: Port 2	USINT	Depends on data type.	---	---

Use *DeviceType* to specify the device type. Set this to _DeviceNXUnit for an NX Unit and _DeviceOptionBoard for an Option Board. The variable used to specify the device is determined by the specified device type.

To specify an NX Unit, use *NxUnit* to specify the device.

In this case, *EcatSlave* and *OptBoard* are not used.

To *NxUnit*, pass the device variable that is assigned to the node location information on the I/O Map for the device to specify.

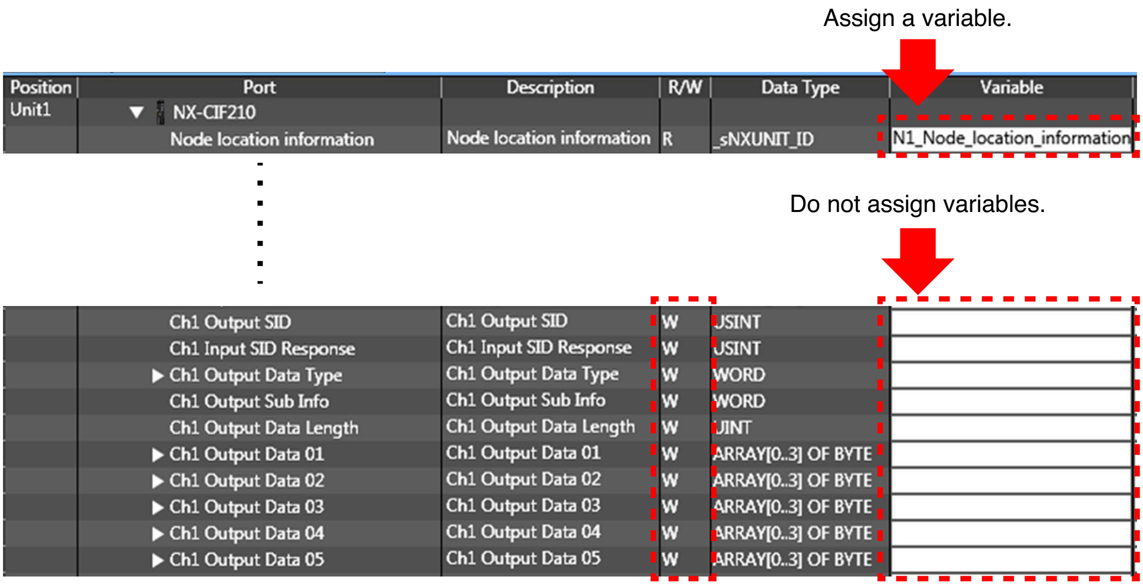
To specify an Option Board, use *OptBoard* to specify the device.

In this case, *NxUnit* and *EcatSlave* are not used.

To *OptBoard*, pass the device variable that is assigned to the node location information on the I/O Map for the device to specify.

If you use this instruction, be sure to assign a device variable to the node location information. Do not assign device variables to any I/O ports following the node location information that are indicated by “W” under the R/W column.

The figure below is an example of using this instruction for port 1 on an NX-CIF210.



Refer to the *Sysmac Studio Version 1 Operation Manual* (Cat. No. W504-E1-07 or later) for details on assigning a device variable to the node location information.

Use *PortNo* to specify the port number.

1: Port 1

2: Port 2

For an NX Unit, set this to Port 1 or Port 2.

For an Option Board, set this to Port 1.

The data type of *DeviceType* is enumerated type _eDEVICE_TYPE.

The meanings of the enumerators of enumerated type _eDEVICE_TYPE are as follows:

Enumerator	Meaning
_DeviceNXUnit	NX Unit is specified.
_DeviceEcatSlave	EtherCAT slave is specified.
_DeviceOptionBoard	Option Board is specified.

In this instruction, you can specify _DeviceNXUnit or _DeviceOptionBoard.

Use the *SlaveAdr* input variable to specify the address of a Modbus-RTU slave.

To broadcast commands to Modbus-RTU slaves, set the *SlaveAdr* input variable to 0.

Use the *WriteCmd* input variable to specify the write command.

CRC is attached by the instruction.

The data type of *WriteCmd* input variable is structure _sSERIAL_MODBUSRTU_WRITE. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
WriteCmd	Write command	Write command	_sSERIAL_MODBUSRTU_WRITE	---	---	---
Fun	Function code	Function code	_eMDB_FUN	_MDB_WRITE_SINGLE_COIL _MDB_WRITE_SINGLE_REGISTER _MDB_WRITE_MULTIPLE_COILS _MDB_WRITE_MULTIPLE_REGISTERS	---	_eMDB_WRITE_SINGLE_COIL
WriteAdr	Write address	Write start address	UINT	Depends on data type.	---	0
WriteSize	Write size	Write size	UINT	Depends on function code.	---	_MDB_WRITE_SINGLE_COIL

The data type of *Fun* is enumerated type _eMDB_FUN.

The meanings of the enumerators of enumerated type _eMDB_FUN are as follows:

Enumerator	Meaning
_MDB_WRITE_SINGLE_COIL	Write an output (bit)
_MDB_WRITE_SINGLE_REGISTER	Write a holding register (word)
_MDB_WRITE_MULTIPLE_COILS	Write multiple outputs (bit)
_MDB_WRITE_MULTIPLE_REGISTERS	Write multiple holding registers (word)

The valid range that you can specify with *WriteSize* varies depending on the function code.

Each value is determined by the size of data that is written and the maximum command length.

The specifications are as follows:

Function code	WriteSize
_MDB_WRITE_SINGLE_COIL	1 (bit)
_MDB_WRITE_SINGLE_REGISTER	1 (word)
_MDB_WRITE_MULTIPLE_COILS	1 to 1,968 (bit)
_MDB_WRITE_MULTIPLE_REGISTERS	1 to 123 (word)

Use the *WriteDat* input variable to specify the data to write.

The data type that you can use for *WriteDat* differs depending on the function code.

The specifications are as follows:

Function code	Data type
_MDB_WRITE_SINGLE_COIL	BOOL BOOL[]
_MDB_WRITE_SINGLE_REGISTER	WORD WORD[]
_MDB_WRITE_MULTIPLE_COILS	BOOL BOOL[]
_MDB_WRITE_MULTIPLE_REGISTERS	WORD WORD[]

To set options, use the *Option* input variable. The specifications are as follows:

The data type of the *Option* input variable is structure _sSERIAL_MODBUS_OPTION. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
Option	Option	Option	_sSERIAL_ MODBUSR-TU_ OPTION	---	---	---
SendDelay	Send delay time	Send delay time	UINT	Depends on data type.	0.01 s	0
TimeOut	Timeout time	2.0 s when the timeout time is set to 0	UINT	Depends on data type.	0.1 s	20
NoResponse	No response	Not used in this instruction.	BOOL	Depends on data type.	---	FALSE
Retry	Retry count	Retry count	USINT	0 to 15	---	0

An error occurs if this instruction is executed for Units other than NX-series Communications Interface Units and Option Boards.

Interruption of Instruction Execution

If *Abort* is changed to TRUE during instruction execution, the execution is interrupted.

When the instruction execution is interrupted, *CommandAborted* changes to TRUE.

If the change of *Abort* is too late to interrupt the execution, *Done* changes to TRUE and the instruction ends normally.

If both *Abort* and *Execute* are changed to TRUE, *CommandAborted* changes to TRUE.

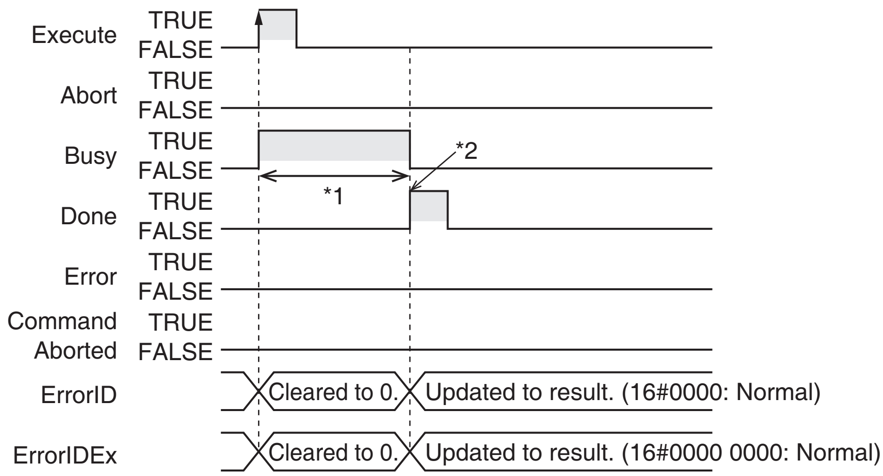
This interruption operation only finishes the *Busy* processing for the instruction, and it does not clear the send or receive buffer. To clear the buffer, use the NX_SerialBufClear instruction.

Timing Charts

The following figures show the timing charts.

● Normal end (when *SendDelay* is 0 (0 s))

The operation is as follows when *SendDelay* is 0 (0 s).

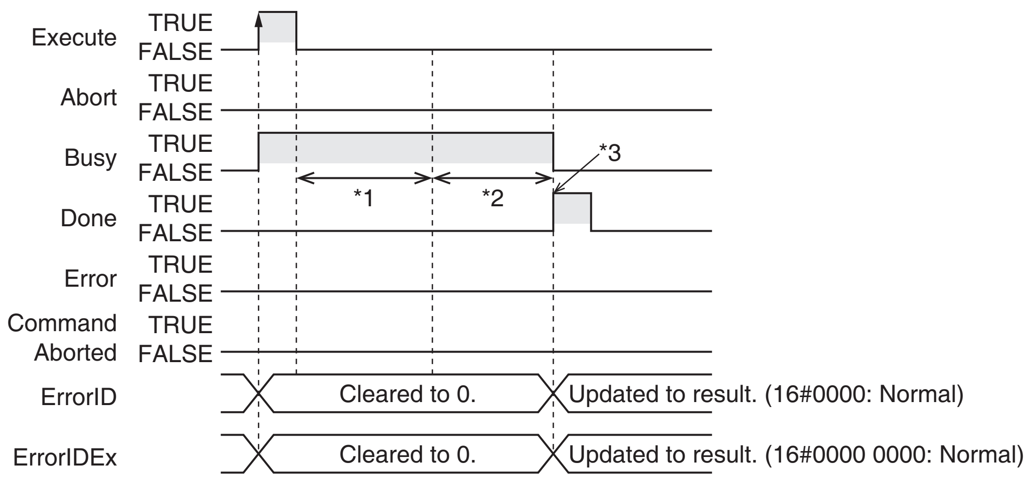


*1 Processing with Modbus-RTU slave

*2 A response to the command is received.

● Normal end (when *SendDelay* is 100 (1 s))

The operation is as follows when *SendDelay* is 100 (1 s).



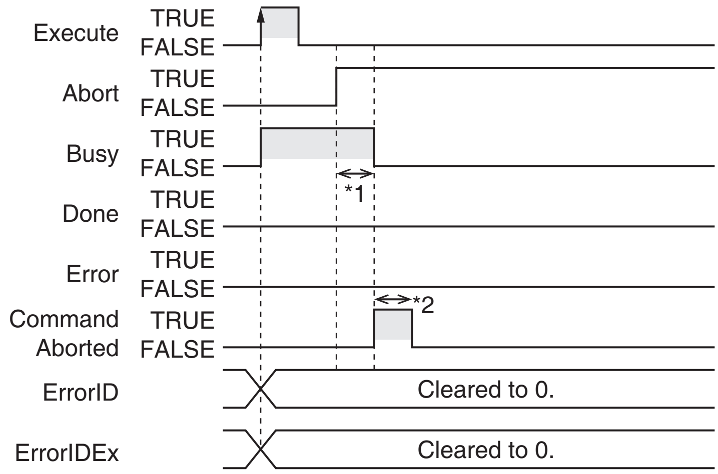
*1 The send delay time of 1 s

*2 A write command is sent to Modbus-RTU slave, and a response is received from Modbus-RTU slave.

*3 A response to the command is received.

● Interruption executed (when *Busy* is TRUE)

The operation is as follows if *Abort* is changed to TRUE while *Busy* is TRUE.

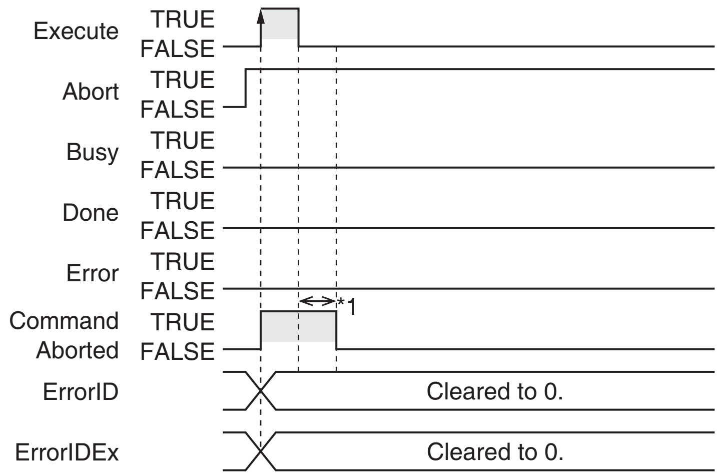


*1 Interruption processing

*2 Changes to FALSE after one task period.

● Interruption executed (when *Execute* is TRUE)

The operation is as follows if both *Abort* and *Execute* are changed to TRUE.



*1 Changes to FALSE after one task period.

Related System-defined Variables

Name	Meaning	Data type	Description
_PLC_OptBoardSta	Option Board Status	ARRAY[1..2] of _sOPTBOARD_STA	• This stores the status of the Option Board.
_NXB_UnitIOActiveTbl	NX Unit I/O Data Active Status	ARRAY[0..8] OF BOOL	• This status tells the NX Units whether I/O data communications can be processed. • The subscript of the array corresponds to the NX Unit numbers. A subscript of 0 means the NX bus master.

Additional Information

Refer to the *MODBUS Application Protocol Specification* for the specifications of the MODBUS communications protocol.

You can obtain *MODBUS Application Protocol Specification* from Modbus Organization, Inc.

http://www.modbus.org/

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing. If *Abort* is changed to TRUE during instruction execution, *CommandAborted* or *Done* changes to TRUE.
- A compiling error will occur if you use this instruction in an event task. Do not use this instruction in event tasks.
- “CIF Unit Initialized” may occur when the NX-series Communications Interface Unit is restarted. Send or receive the data again, if necessary.
- If you use this instruction, do not assign device variables to any I/O ports that are indicated by “W” under the R/W column on the I/O Map Tab Page in the Sysmac Studio for the applicable NX-series Communications Interface Unit.
- Data may still remain in the buffer of the target device port in the following cases. To clear the buffer, execute the NX_SerialBufClear instruction before executing the following instruction: NX_ModbusRtuCmd instruction, NX_ModbusRtuRead instruction, or NX_ModbusRtuWrite instruction.
    - After the operation starts or when you change the operating mode to RUN mode.
    - The retry was set (i.e., *Option.Retry* is not 0) in the previous instruction execution.
    - The previous instruction execution is interrupted (i.e., the *CommandAborted* output variable is TRUE).
    - An error occurred (i.e., *Error* is TRUE) in the previous instruction execution.
- An error will occur in the following cases. *Error* will change to TRUE.
    - A value that is out of range was set for *SlaveAdr*, *WriteCmd.Fun*, *WriteCmd.WriteSize*, *Option.Retry*, *DevicePort.DevicePortType*, or *DevicePort.PortNo*.
    - The variable specified with *WriteDat* is smaller than the size specified with *WriteCmd.WriteSize*.
    - The Unit or port specified with *DevicePort* does not exist.
    - The data type of *DevicePort* or *WriteDat* is invalid.
    - If more than 32 instructions from the NX_SerialSend instruction, NX_SerialRcv instruction, NX_ModbusRtuCmd instruction, NX_ModbusRtuRead instruction, NX_ModbusRtuWrite instruction, NX_SerialSigCtl instruction, NX_SerialSigRead instruction, NX_SerialStatusRead instruction, NX_SerialBufClear instruction, NX_SerialStartMon instruction and NX_SerialStopMon instruction are executed at the same time.

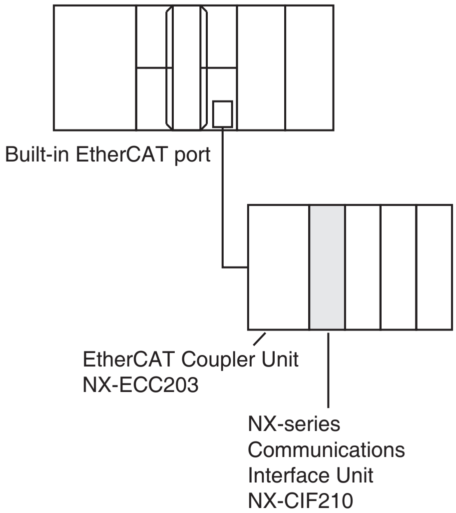
  - This instruction is executed with a device port variable that is the same as the one specified for the instruction which is still being executed. In this case, the instruction which is still being executed is one of the followings.
    The NX_SerialSend instruction, NX_SerialRcv instruction, NX_ModbusRtuCmd instruction, NX_-ModbusRtuRead instruction, and NX_ModbusRtuWrite instruction.
  - A parity error occurred in the data received.
  - A framing error occurred in the data received.
  - An overrun error occurred in the data received.
  - CRC mismatch occurred for the received data.
  - Timeout time elapsed.
  - This instruction is executed for Units other than NX-series Communications Interface Units and Option Boards.
  - An Exception Response was received from a Modbus-RTU slave. You can check Exception Codes with the *ErrorIDEx* output variable.
  - There was an invalid function code, receive size, etc. in the response data from a Modbus-RTU slave.
  - The serial communications mode of the specified Option Board is not *Modbus-RTU master*.
- In this instruction, the expansion error code *ErrorIDEx* is displayed when an error is detected in a Modbus-RTU slave. An expansion error code is output to *ErrorIDEx* when the value of error code *ErrorID* is WORD#16#0C10. The display format is ErrorIDEx=000000XX. For the value XX, refer to the Exception Code specifications of the MODBUS communications protocol.
  Refer to the *MODBUS Application Protocol Specification* for the Exception Code specifications of the MODBUS communications protocol.
  You can obtain *MODBUS Application Protocol Specification* from Modbus Organization, Inc.
  http://www.modbus.org/

Sample Programming

In this sample, an NX-series Communications Interface Unit (NX-CIF210) is connected to an EtherCAT Coupler Unit (NX-ECC203).

The unit number of the NX-CIF210 is set to 1.

For the Unit operation settings of the NX-CIF210, set **Ch2 Number of Characters to Determine the End** to 35. The number of characters is regarded as 3.5 during operation because the unit for setting the Number of Characters to Determine the End is 0.1 character.



When *Trigger* changes to TRUE, the instruction clears the buffer of the serial port 2 on the NX-CIF210 and then sends a Modbus-RTU command.

It changes an output from the write start address 149 in slave address 1.

Write commands are sent/received to write a variable.

Internal Variables	Variable	Data type	Initial value	Comment
	Stage	INT	0	
	Trigger	BOOL	FALSE	Execution condition
	DevicePort	_sDEVICE_PORT		Port settings
	NX_SerialBufClear_instance	NX_SerialBufClear		Clear buffer
	ClearDone	BOOL		
	ClearError	BOOL		
	NX_ModbusRtuWrite_instance	NX_ModbusRtuWrite		
	ModbusSlaveAdr	UINT	UINT#0	Slave address
	ModbusDone	BOOL		
	ModbusCommandAborted	BOOL		
	ModbusError	BOOL		
	DoModbusTrigger	BOOL		
	ModbusWriteDat	ARRAY[0..5] OF BOOL	[6(FALSE)]	
	ModbusWriteCmd	_sSERIAL_MODBUSRTU_WRITE		

Serial Communications Instructions

2

NX_ModbusRtuWrite

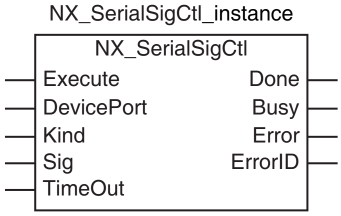
External Variables	Variable	Data type	Constant	Comment
	N1_Node_location_information	_sNXUNIT_ID	✔	

```
// Start sequence when Trigger changes to TRUE.
IF (Trigger=TRUE) AND (DoModbusTrigger=FALSE) THEN
    DoModbusTrigger := TRUE;

    NX_SerialBufClear_instance(Execute := FALSE,
        DevicePort:=DevicePort);
    NX_ModbusRtuWrite_instance(Execute:= FALSE,
        DevicePort:=DevicePort,
        WriteDat:=ModbusWriteDat);
    Stage := 1;        // Initialization completed.
END_IF;

IF (DoModbusTrigger=TRUE) THEN
    CASE Stage OF
    1:    // Buffer clear request
        DevicePort.DeviceType:=_eDEVICE_TYPE#_DeviceNXUnit;
        DevicePort.NxUnit:=N1_Node_location_information;
        DevicePort.PortNo:=2;

        NX_SerialBufClear_instance(Execute := TRUE,
            DevicePort:=DevicePort,
            Done => ClearDone,
            Error => ClearError);

        IF (ClearDone = TRUE) THEN
            Stage := 2;        // Buffer clear is normal end.
        ELSIF (ClearError = TRUE) THEN
            Stage := 99;        // Buffer clear is error end.
        END_IF;

    2:    /// Modbus  write request
        ModbusSlaveAdr := 1;                             // Slave address
        ModbusWriteCmd.Fun:=_MDB_WRITE_SINGLE_COIL;      // Function code
        ModbusWriteCmd.WriteAdr:=149;                    // Write address
        ModbusWriteCmd.WriteSize:=1;                     // Write size

        NX_ModbusRtuWrite_instance(Execute:= TRUE,
            DevicePort:=DevicePort,
            SlaveAdr:=ModbusSlaveAdr,
            WriteCmd:=ModbusWriteCmd,
            WriteDat:=ModbusWriteDat,
            Done=>ModbusDone,
            CommandAborted=>ModbusCommandAborted,
            Error=>ModbusError);

        IF (ModbusDone = TRUE) THEN
            Stage := 3;        // The NX_ModbusRtuWrite instruction is normal end.
        ELSIF (ModbusError=TRUE) OR (ModbusCommandAborted=TRUE) THEN
            Stage :=99;        // The NX_ModbusRtuWrite instruction is error end or
Abort.
        END_IF;

    3:    // Processing after the NX_ModbusRtuWrite instruction is normal end.
        Trigger := FALSE;
```

```
        DoModbusTrigger := FALSE;

    99:    // Error Processing
        Trigger := FALSE;
        DoModbusTrigger := FALSE;
    END_CASE;
END_IF;
```

NX_SerialSigCtl

The NX_SerialSigCtl instruction turns ON or OFF the ER or RS signal of a serial port on an NX-series Communications Interface Unit or Option Board.

Instruction	Name	FB/FUN	Graphic expression	ST expression
NX_Serial-SigCtl	Serial Control Signal ON/OFF Switching	FB		NX_SerialSigCtl_instance(Execute, DevicePort, Kind, Sig, TimeOut, Done, Busy, Error, ErrorID);

Version Information

A CPU Unit with unit version 1.11 or later and Sysmac Studio version 1.15 or higher are required to use this instruction.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
DevicePort	Device port	Input	Object that represents a device port	---	---	---
Kind	Signal command		Signal command	_RS_SIG _ER_SIG*1	---	*2
Sig	ON/OFF command		ON/OFF command	Depends on data type.	---	*2
TimeOut	Timeout time		2.0 s when the timeout time is set to 0	Depends on data type.	0.1 s	0

*1 You cannot use _CS_SIG or _DR_SIG. If either of them is specified, an error will occur when the instruction is executed.

*2 If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
DevicePort	Refer to *Function* for details on the structure _sDEVICE_PORT.																			
Kind	Refer to *Function* for the enumerators of the enumerated type _eSERIAL_SIG.																			
Sig	OK																			
TimeOut							OK													

Function

The NX_SerialSigCtl instruction turns ON or OFF the ER or RS signal of a serial port on an NX-series Communications Interface Unit or Option Board.

The data type of the *DevicePort* input variable is structure _sDEVICE_PORT. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
DevicePort	Device port	Object that represents a device port	_sDE-VICE_PORT	---	---	---
DeviceType	Device type	Type of the device to specify	_eDEVICE_-TYPE	_DeviceNXUnit _DeviceEcat-Slave _DeviceOption-Board	---	---
NxUnit	Specified Unit	NX Unit to control	_sNXUNIT_ID	---	---	---
EcatSlave	Specified slave	EtherCAT slave to con-trol	_sECAT_ID	---	---	---
OptBoard	Specified Option Board	Option Board to control	_sOPTBOAR D_ID	---	---	---
Reserved	Reserved	Reserved	Reserved	---	---	---
PortNo	Port number	Port number 1: Port 1 2: Port 2	USINT	Depends on data type.	---	---

Use *DeviceType* to specify the device type. Set this to _DeviceNXUnit for an NX Unit and _DeviceOp-tionBoard for an Option Board. The variable used to specify the device is determined by the specified device type.

To specify an NX Unit, use *NxUnit* to specify the device.

In this case, *EcatSlave* and *OptBoard* are not used.

To *NxUnit*, pass the device variable that is assigned to the node location information on the I/O Map for the device to specify.

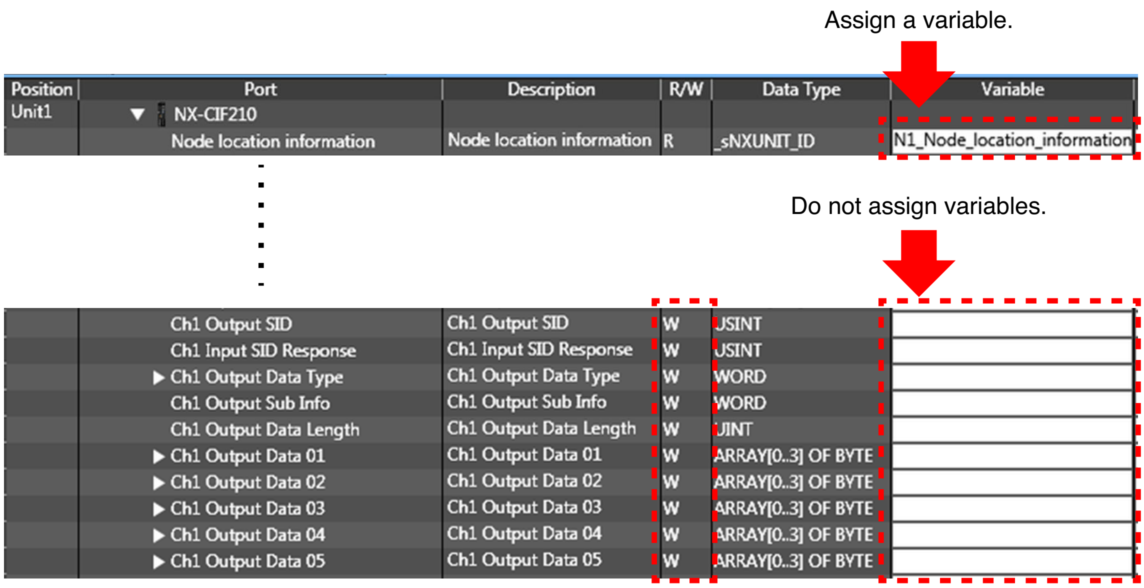
To specify an Option Board, use *OptBoard* to specify the device.

In this case, *NxUnit* and *EcatSlave* are not used.

To *OptBoard*, pass the device variable that is assigned to the node location information on the I/O Map for the device to specify.

If you use this instruction, be sure to assign a device variable to the node location information. Do not assign device variables to any I/O ports following the node location information that are indicated by “W” under the R/W column.

The figure below is an example of using this instruction for port 1 on an NX-CIF210.



Refer to the *Sysmac Studio Version 1 Operation Manual* (Cat. No. W504-E1-07 or later) for details on assigning a device variable to the node location information.

Use *PortNo* to specify the port number.

1: Port 1

2: Port 2

For an NX Unit, set this to Port 1 or Port 2.

For an Option Board, set this to Port 1.

The data type of *DeviceType* is enumerated type _eDEVICE_TYPE.

The meanings of the enumerators of enumerated type _eDEVICE_TYPE are as follows:

Enumerator	Meaning
_DeviceNXUnit	NX Unit is specified.
_DeviceEcatSlave	EtherCAT slave is specified.
_DeviceOptionBoard	Option Board is specified.

In this instruction, you can specify _DeviceNXUnit or _DeviceOptionBoard.

Use the *Kind* input variable to select the ER or RS signal.

When the *Sig* input variable is TRUE, the ER or RS signal turns ON.

When the *Sig* input variable is FALSE, the ER or RS signal turns OFF.

The data type of *Kind* is enumerated type _eSERIAL_SIG.

The meanings of the enumerators of enumerated type _eSERIAL_SIG are as follows:

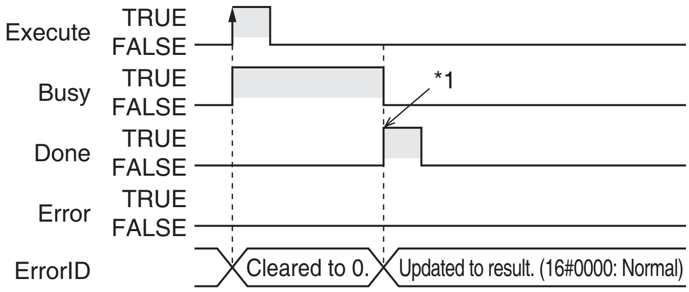
Enumerator	Meaning
_RS_SIG	RS signal
_ER_SIG	ER signal
_CS_SIG	CS signal
_DR_SIG	DR signal

An error occurs if this instruction is executed for Units other than NX-series Communications Interface Units and Option Boards.

Timing Charts

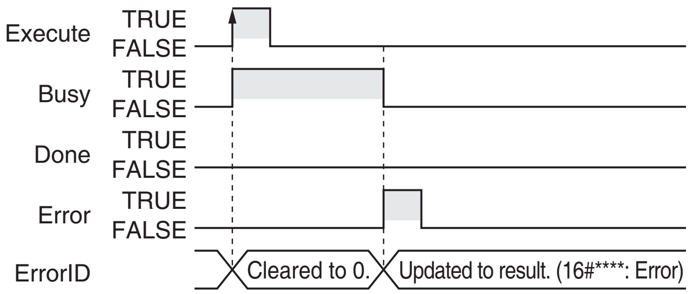
The following figures show the timing charts.

● Normal end



*1 Signal ON/OFF control is completed.

● Error end



Related System-defined Variables

Name	Meaning	Data type	Description
_PLC_OptBoardSta	Option Board Status	ARRAY[1..2] of _sOPTBOARD_STA	• This stores the status of the Option Board.
_NXB_UnitIOActiveTbl	NX Unit I/O Data Active Status	ARRAY[0..8] OF BOOL	• This status tells the NX Units whether I/O data communications can be processed. • The subscript of the array corresponds to the NX Unit numbers. A subscript of 0 means the NX bus master.

Precautions for Correct Use

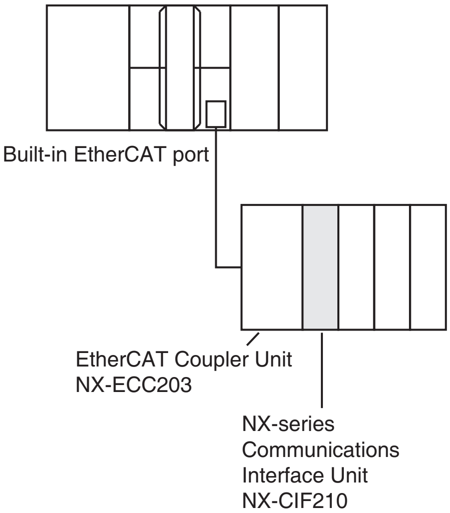
- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- A compiling error will occur if you use this instruction in an event task. Do not use this instruction in event tasks.
- This instruction does not check the communications protocol and wiring conditions. Check the wiring conditions and communications protocol before you use this instruction.

- “CIF Unit Initialized” may occur when the NX-series Communications Interface Unit is restarted. Send or receive the data again, if necessary.
- If you use this instruction, do not assign device variables to any I/O ports that are indicated by “W” under the R/W column on the I/O Map Tab Page in the Sysmac Studio for the applicable NX-series Communications Interface Unit.
- An error will occur in the following cases. *Error* will change to TRUE.
  - A value that is out of range was set for *Kind*, *DevicePort.DevicePortType*, or *DevicePort.PortNo*.
  - The Unit or port specified with *DevicePort* does not exist.
  - If an RS-422A/485 serial port is specified for *DevicePort.*
  - When **RS/CS flow control** is selected for the flow control setting of the NX-series Communications Interface Unit and this instruction sends “RS Signal ON” or “RS Signal OFF”.
  - If more than 32 instructions from the NX_SerialSend instruction, NX_SerialRcv instruction, NX_-ModbusRtuCmd instruction, NX_ModbusRtuRead instruction, NX_ModbusRtuWrite instruction, NX_SerialSigCtl instruction, NX_SerialSigRead instruction, NX_SerialStatusRead instruction, NX_SerialBufClear instruction, NX_SerialStartMon instruction and NX_SerialStopMon instruction are executed at the same time.
  - This instruction is executed with a device port variable that is the same as the one specified for the instruction which is still being executed. In this case, the instruction which is still being executed is one of the followings.
    The NX_SerialSigRead instruction, NX_SerialStatusRead instruction, NX_SerialSigCtl instruction, NX_SerialBufClear instruction, NX_SerialStartMon instruction, and NX_SerialStopMon instruction.
  - Timeout time elapsed during serial communications.
  - This instruction is executed for Units other than NX-series Communications Interface Units and Option Boards.
  - The serial communications mode of the specified Option Board is not *No-protocol* or *Modbus-RTU master*.

Sample Programming

In this sample, an NX-series Communications Interface Unit (NX-CIF210) is connected to an EtherCAT Coupler Unit (NX-ECC203).

The unit number of the NX-CIF210 is set to 1.



The ER signal is turned ON if the SetER signal is turned ON for a no-protocol remote node that is connected to serial port 2 of the NX-CIF210. The ER signal is turned OFF if the ResetER signal is turned ON for the same remote node.

Definitions of Global Variables

Global Variables

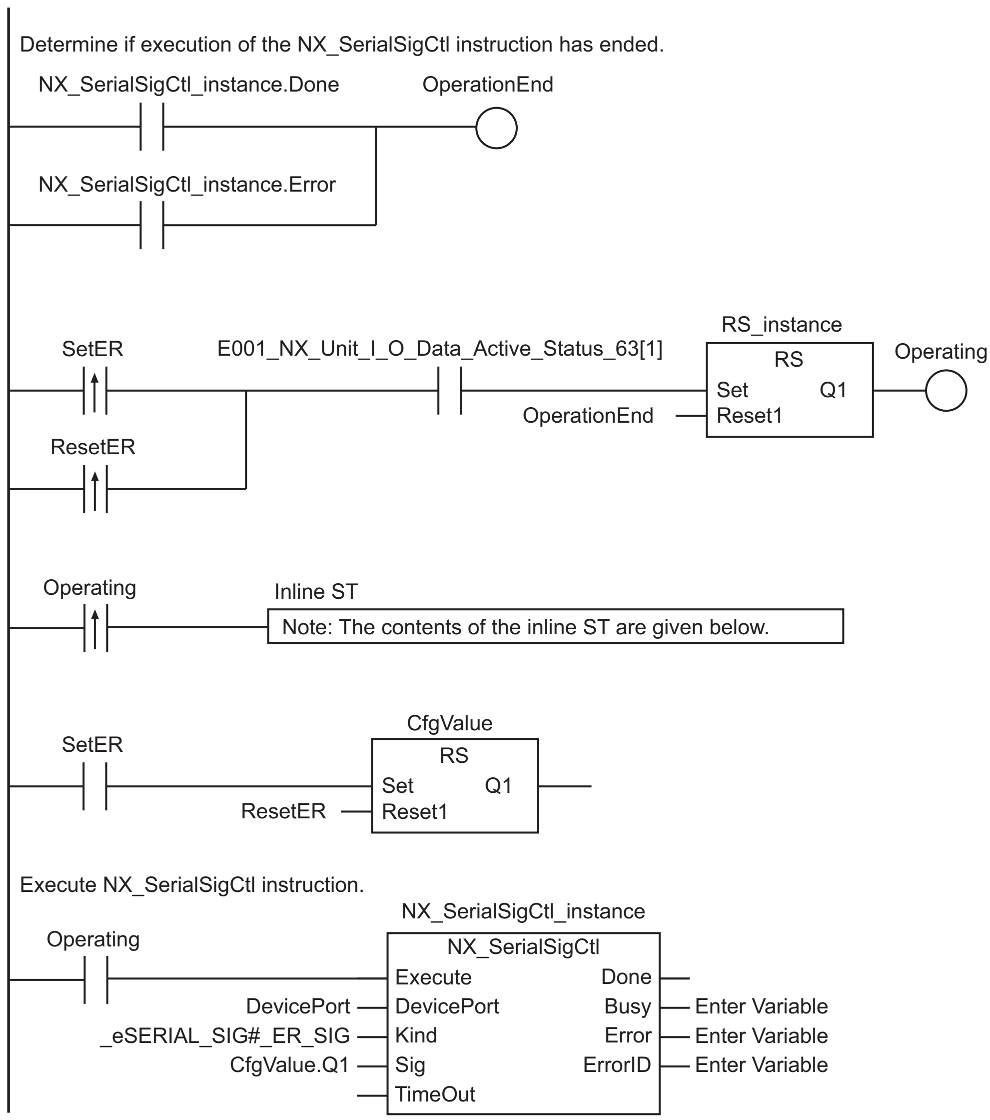
Name	Data type	AT	Comment
E001_NX_Unit_I_O_Data_Active_Status_63	ARRAY[0..63] OF BOOL	ECAT://node#1/NX Unit I/O Data Active Status 125	Usage of I/O data for 63 NX Units.
N1_Node_location_information	_sNXUNIT_ID	---	Device variable to specify NX-CIF210*1

*1 On the Sysmac Studio, right-click an NX-series slave terminal unit, select **Display Node Location Port**, and set the device variable. Refer to the *Sysmac Studio Version 1 Operation Manual* (Cat. No. W504-E1-07 or later) for details.

LD

Internal Variables	Variable	Data type	Initial value	Comment
	OperationEnd	BOOL	FALSE	Processing completed
	SetER	BOOL	FALSE	ER signal ON execution condition
	ResetER	BOOL	FALSE	ER signal OFF execution condition
	Operating	BOOL	FALSE	Processing
	DevicePort	_sDEVICE_PORT		Port settings
	RS_instance	RS	---	*Operating* retained
	CfgValue	RS	---	Value determined by *SetER* or *ResetER*
	NX_SerialSigCtl_instance	NX_SerialSigCtl	---	

External Variables	Variable	Data type	Comment
	E001_NX_Unit_I_O_Data_Active_Status_63	ARRAY[0..63] OF BOOL	• Usage of I/O data for 63 NX Units. • If the relevant Unit number is 1, E001_NX_Unit_I_O_Data_Active_Status_63[1] is used.
	N1_Node_location_information	_sNXUNIT_ID	Device variable to specify NX-CIF210



● Contents of Inline ST

```
DevicePort.DeviceType:=_eDEVICE_TYPE#_DeviceNXUnit;
DevicePort.NxUnit:=N1_Node_location_information;
DevicePort.PortNo:=2;
```

ST

Internal Variables	Variable	Data type	Initial value	Comment
	OperatingStart	BOOL	FALSE	Processing started
	SetER	BOOL	FALSE	ER signal ON execution condition
	ResetER	BOOL	FALSE	ER signal OFF execution condition
	DevicePort	_sDEVICE_PORT		Port settings
	CfgValue	RS	---	Value determined by *SetER* or *ResetER*
	NX_SerialSigCtl_instance	NX_SerialSigCtl	---	

External Variables	Name	Data type	Comment
	E001_NX_Unit_I_O_Data_Active_Status_63	ARRAY[0..63] OF BOOL	• Usage of I/O data for 63 NX Units. • If the relevant Unit number is 1, E001_NX-_Unit_I_O_Data_Active_Status_63[1] is used.
	N1_Node_location_information	_sNXUNIT_ID	Device variable to specify NX-CIF210

```
// Detection of SetER or ResetER
IF (NX_SerialSigCtl_instance.Done OR NX_SerialSigCtl_instance.Error) THEN
         OperatingStart:=FALSE;
ELSE_IF
         OperatingStart:=(SetER OR ResetER)
                                        AND E001_NX_Unit_I_O_Data_Active_Sta-
tus_63[1]
                                        AND NOT(P_FirstRun);
         DevicePort.DeviceType:=_eDEVICE_TYPE#_DeviceNXUnit;
         DevicePort.NxUnit:=N1_Node_location_information;
         DevicePort.PortNo:=2;
END_IF;

// ER signal value is determined.
CfgValue(Set:=SetER, Reset1:=ResetER);

// NX_SerialSigCtl instruction is executed.
NX_SerialSigCtl_instance(Execute:=OperatingStart,
                         DevicePort:=DevicePort,
                         Kind:=_eSERIAL_SIG#_SIG_ER,
                         Sig:=CfgValue.Q1);
```

NX_SerialBufClear

The NX_SerialBufClear instruction clears the send or receive buffer.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
NX_SerialBuf-Clear	Clear Buffer	FB	NX_SerialBufClear_instance NX_SerialBufClear Execute — Done DevicePort — Busy BufKind — Error TimeOut — ErrorID	NX_SerialBufClear_instance(Execute DevicePort, BufKind, TimeOut, Done, Busy, Error, ErrorID);

Version Information

A CPU Unit with unit version 1.11 or later and Sysmac Studio version 1.15 or higher are required to use this instruction.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
DevicePort	Device port	Input	Object that represents a device port	---	---	---
BufKind	Buffer type		Type (send or receive) of buffer	_BUF_SENDRCV _BUF_SEND _BUF_RCV	---	_BUF-SEN-DRCV
TimeOut	Timeout time		2.0 s when the timeout time is set to 0	Depends on data type.	0.1 s	0

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
DevicePort	Refer to *Function* for details on the structure _sDEVICE_PORT.																			
BufKind	Refer to *Function* for the enumerators of the enumerated type _eSERIAL_BUF_KIND.																			
TimeOut							OK													

Function

The NX_SerialBufClear clears data in a buffer according to the setting of type of the port and buffer. The instruction ends normally when the clear processing is completed.

The data type of the *DevicePort* input variable is structure _sDEVICE_PORT. The specifications are as follows:

Name	Meaning	Description	Data type	Valid range	Unit	Default
DevicePort	Device port	Object that represents a device port	_sDE-VICE_PORT	---	---	---
DeviceType	Device type	Type of the device to specify	_eDEVICE_-TYPE	_DeviceNXUnit _DeviceEcat-Slave _DeviceOption-Board	---	---
NxUnit	Specified Unit	NX Unit to control	_sNXUNIT_ID	---	---	---
EcatSlave	Specified slave	EtherCAT slave to con-trol	_sECAT_ID	---	---	---
OptBoard	Specified Option Board	Option Board to control	_sOPT-BOARD_ID	---	---	---
Reserved	Reserved	Reserved	Reserved	---	---	---
PortNo	Port number	Port number 1: Port 1 2: Port 2	USINT	Depends on data type.	---	---

Use *DeviceType* to specify the device type. Set this to _DeviceNXUnit for an NX Unit and _DeviceOp-tionBoard for an Option Board. The variable used to specify the device is determined by the specified device type.

To specify an NX Unit, use *NxUnit* to specify the device.

In this case, *EcatSlave* and *OptBoard* are not used.

To *NxUnit*, pass the device variable that is assigned to the node location information on the I/O Map for the device to specify.

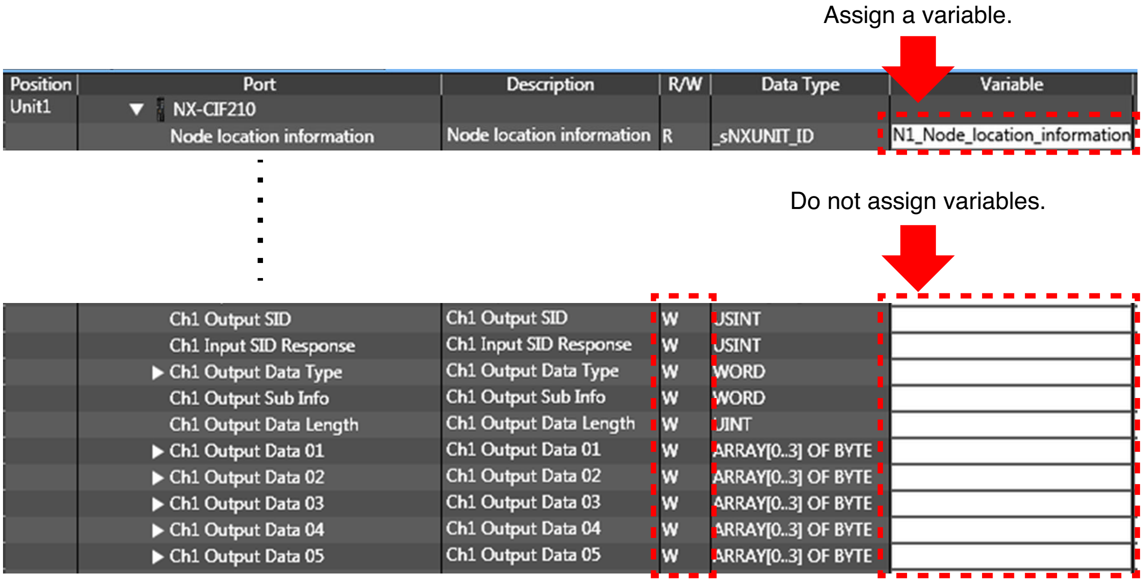
To specify an Option Board, use *OptBoard* to specify the device.

In this case, *NxUnit* and *EcatSlave* are not used.

To *OptBoard*, pass the device variable that is assigned to the node location information on the I/O Map for the device to specify.

If you use this instruction, be sure to assign a device variable to the node location information. Do not assign device variables to any I/O ports following the node location information that are indicated by “W” under the R/W column.

The figure below is an example of using this instruction for port 1 on an NX-CIF210.



Refer to the *Sysmac Studio Version 1 Operation Manual* (Cat. No. W504-E1-07 or later) for details on assigning a device variable to the node location information.

Use *PortNo* to specify the port number.

1: Port 1

2: Port 2

For an NX Unit, set this to Port 1 or Port 2.

For an Option Board, set this to Port 1.

The data type of *DeviceType* is enumerated type _eDEVICE_TYPE.

The meanings of the enumerators of enumerated type _eDEVICE_TYPE are as follows:

Enumerator	Meaning
_DeviceNXUnit	NX Unit is specified.
_DeviceEcatSlave	EtherCAT slave is specified.
_DeviceOptionBoard	Option Board is specified.

In this instruction, you can specify _DeviceNXUnit or _DeviceOptionBoard.

Specify the port with *Port*, and specify the buffer to clear with *BufKind*.

Data is not cleared if it is the data that the NX-series Communications Interface Unit received from the external devices after the receive buffer is cleared.

The data type of *BufKind* is enumerated type _eSERIAL_BUF_KIND.

The meanings of the enumerators of enumerated type _eSERIAL_BUF_KIND are as follows:

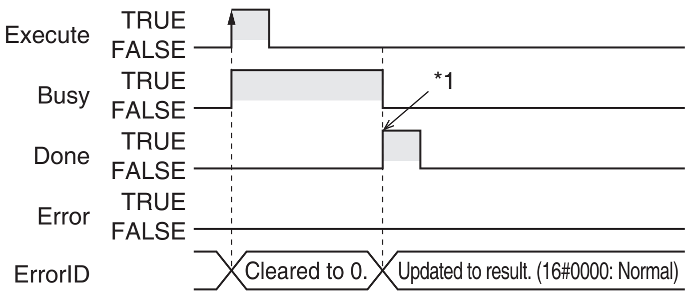
Enumerator	Meaning
_BUF_SENDRCV	Send buffer and receive buffer
_BUF_SEND	Send buffer
_BUF_RCV	Receive buffer

An error occurs if this instruction is executed for Units other than NX-series Communications Interface Units and Option Boards.

Timing Charts

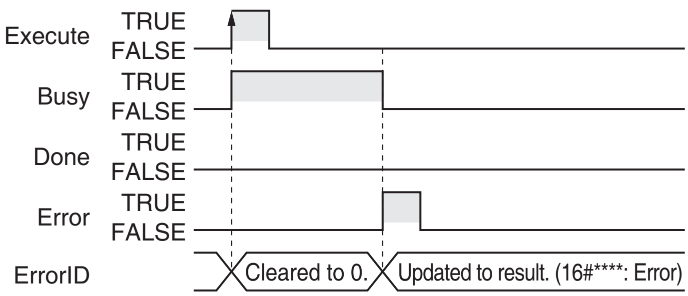
The following figures show the timing charts.

● Normal end



*1 Buffer clear processing is completed.

● Error end



Related System-defined Variables

Name	Meaning	Data type	Description
_PLC_OptBoardSta	Option Board Status	ARRAY[1..2] of _sOPTBOARD_STA	• This stores the status of the Option Board.
_NXB_UnitIOActiveTbl	NX Unit I/O Data Active Status	ARRAY[0..8] OF BOOL	• This status tells the NX Units whether I/O data communications can be processed. • The subscript of the array corresponds to the NX Unit numbers. A subscript of 0 means the NX bus master.

Precautions for Correct Use

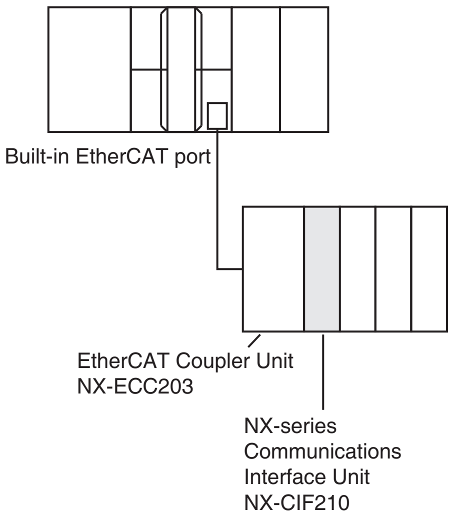
- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- A compiling error will occur if you use this instruction in an event task. Do not use this instruction in event tasks.

- This instruction does not check the communications protocol and wiring conditions. Check the wiring conditions and communications protocol before you use this instruction.
- “CIF Unit Initialized” may occur when the NX-series Communications Interface Unit is restarted. Send or receive the data again, if necessary.
- If you use this instruction, do not assign device variables to any I/O ports that are indicated by “W” under the R/W column on the I/O Map Tab Page in the Sysmac Studio for the applicable NX-series Communications Interface Unit.
- An error will occur in the following cases. *Error* will change to TRUE.
  - A value that is out of range was set for *BufKind*, *DevicePort.DevicePortType*, or *Device-Port.PortNo*.
  - The Unit, Option Board, or port specified with *DevicePort* does not exist.
  - If more than 32 instructions from the NX_SerialSend instruction, NX_SerialRcv instruction, NX_-ModbusRtuCmd instruction, NX_ModbusRtuRead instruction, NX_ModbusRtuWrite instruction, NX_SerialSigCtl instruction, NX_SerialSigRead instruction, NX_SerialStatusRead instruction, NX_SerialBufClear instruction, NX_SerialStartMon instruction and NX_SerialStopMon instruction are executed at the same time.
  - This instruction is executed with a device port variable that is the same as the one specified for the instruction which is still being executed. In this case, the instruction which is still being executed is one of the followings.
    The NX_SerialSend instruction, NX_SerialRcv instruction, NX_ModbusRtuCmd instruction, NX_-ModbusRtuRead instruction, NX_ModbusRtuWrite instruction, NX_SerialSigCtl instruction, NX_SerialSigRead instruction, NX_SerialStatusRead instruction, NX_SerialBufClear instruction, NX_SerialStartMon instruction, and NX_SerialStopMon instruction.
  - Timeout time elapsed during serial communications.
  - This instruction is executed for Units other than NX-series Communications Interface Units and Option Boards.
  - The serial communications mode of the specified Option Board is not *No-protocol* or *Modbus-RTU master*.

Sample Programming

In this sample, an NX-series Communications Interface Unit (NX-CIF210) is connected to an EtherCAT Coupler Unit (NX-ECC203).

The unit number of the NX-CIF210 is set to 1.



This instruction clears the receive buffer of serial port 2 on NX-CIF210. When clear processing is completed, the instruction waits for data that does not have start code and has the CR end code.

Definitions of Global Variables

Global Variables

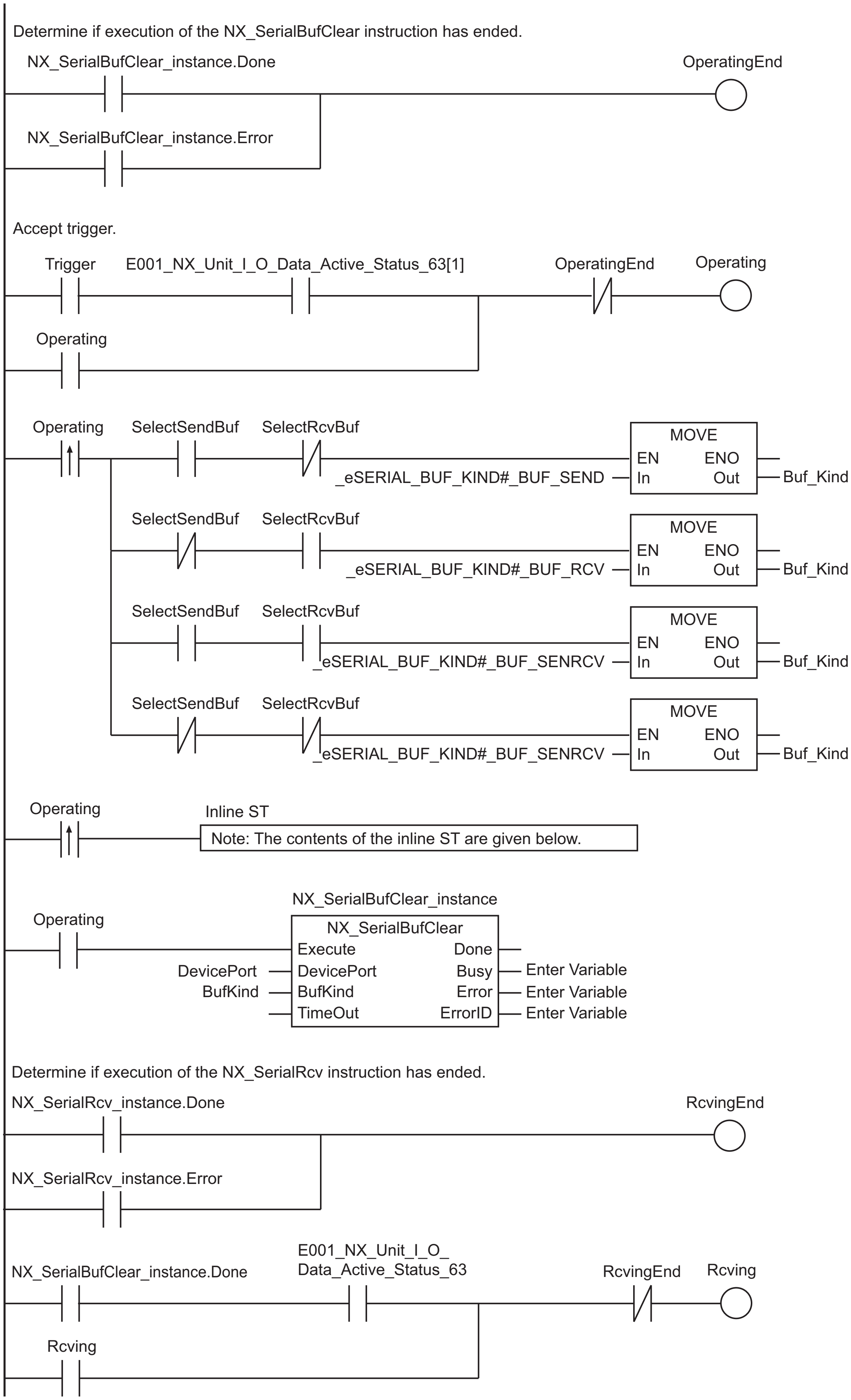
Name	Data type	AT	Comment
E001_NX_Unit_I_O_Data_Active_Status_63	ARRAY[0..63] OF BOOL	ECAT://node#1/NX Unit I/O Data Active Status 125	Usage of I/O data for 63 NX Units.
N1_Node_location_information	_sNXUNIT_ID	---	Device variable to specify NX-CIF210*1

*1 On the Sysmac Studio, right-click an NX-series slave terminal unit, select **Display Node Location Port,** and set the device variable. Refer to the *Sysmac Studio Version 1 Operation Manual* (Cat. No. W504-E1-07 or later) for details.

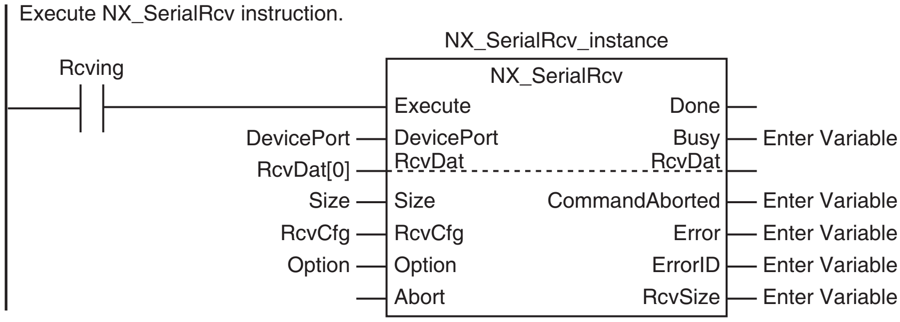
LD

Internal Variables	Variable	Data type	Initial value	Comment
	OperatingEnd	BOOL	FALSE	Buffer clear processing finished
	Trigger	BOOL	FALSE	Buffer clear execution condition
	Operating	BOOL	FALSE	Buffer clear processing in progress
	SelectSendBuf	BOOL	FALSE	Send buffer selection
	SelectRcvBuf	BOOL	FALSE	Receive buffer selection
	BufKind	_eSERIAL_BUF_KIND	_BUF_SENDRCV	Buffer setting
	DevicePort	_sDEVICE_PORT		Port settings
	NX_SerialBufClear_instance	NX_SerialBufClear	---	
	RcvingEnd	BOOL		Receive processing completed
	Rcving	BOOL		Receive processing in progress
	RcvCfg	_sSERIAL_CFG		Reception completion setting
	StartTrig	_eSERIAL_START	_SERIAL_START_NONE	
	StartCode	ARRAY[0..1] OF BYTE	[2(16#0)]	
	EndTrig	_eSERIAL_END	_SERIAL_END_CODE1	
	EndCode	ARRAY[0..1] OF BYTE	[16#0D,16#00]	End code: CR
	RcvSizeCfg	UINT	0	
	Option	_sSERIAL_RCV_OPTION		
	TimeOut	TIME	TIME#0s	
	LastDatRcv	BOOL	FALSE	
	ClearBuf	BOOL	FALSE	

External Variables	Variable	Data type	Comment
	E001_NX_Unit_I_O_Data_Active_Status_63	ARRAY[0..63] OF BOOL	• Usage of I/O data for 63 NX Units. • If the relevant Unit number is 1, E001_NX_Unit_I_O_Data_Active_Status_63[1] is used.
	N1_Node_location_information	_sNXUNIT_ID	Device variable to specify NX-CIF210



Determine if execution of the NX_SerialBufClear instruction has ended.
NX_SerialBufClear_instance.Done
OperatingEnd
NX_SerialBufClear_instance.Error
Accept trigger.
Trigger
E001_NX_Unit_I_O_Data_Active_Status_63[1]
OperatingEnd
Operating
Operating
Operating
SelectSendBuf
SelectRcvBuf
MOVE
EN
ENO
_eSERIAL_BUF_KIND#_BUF_SEND
In
Out
Buf_Kind
SelectSendBuf
SelectRcvBuf
MOVE
EN
ENO
_eSERIAL_BUF_KIND#_BUF_RCV
In
Out
Buf_Kind
SelectSendBuf
SelectRcvBuf
MOVE
EN
ENO
_eSERIAL_BUF_KIND#_BUF_SENRCV
In
Out
Buf_Kind
SelectSendBuf
SelectRcvBuf
MOVE
EN
ENO
_eSERIAL_BUF_KIND#_BUF_SENRCV
In
Out
Buf_Kind
Operating
Inline ST
Note: The contents of the inline ST are given below.
NX_SerialBufClear_instance
Operating
NX_SerialBufClear
Execute
Done
DevicePort
DevicePort
Busy
Enter Variable
BufKind
BufKind
Error
Enter Variable
TimeOut
ErrorID
Enter Variable
Determine if execution of the NX_SerialRcv instruction has ended.
NX_SerialRcv_instance.Done
RcvingEnd
NX_SerialRcv_instance.Error
NX_SerialBufClear_instance.Done
E001_NX_Unit_I_O_
Data_Active_Status_63
RcvingEnd
Rcving
Rcving



● Contents of Inline ST

```
DevicePort.DeviceType:=_eDEVICE_TYPE#_DeviceNXUnit;
DevicePort.NxUnit:=N1_Node_location_information;
DevicePort.PortNo:=2;
```

ST

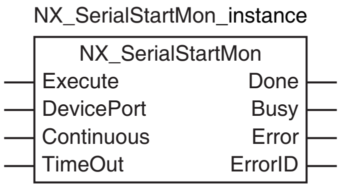
Internal Variables	Variable	Data type	Initial value	Comment
	OperatingEnd	BOOL	FALSE	Buffer clear processing finished
	Trigger	BOOL	FALSE	Buffer clear execution condition
	Operating	BOOL	FALSE	Buffer clear processing in progress
	SelectSendBuf	BOOL	FALSE	Send buffer selection
	SelectRcvBuf	BOOL	FALSE	Receive buffer selection
	BufKind	_eSERIAL_BUF_KIND	_BUF_SENDRCV	Buffer setting
	DevicePort	_sDEVICE_PORT		Port settings
	NX_SerialBuf-Clear_instance	NX_SerialBufClear	---	
	RcvingEnd	BOOL		Receive processing com-pleted
	Rcving	BOOL		Receive processing in progress
	RcvCfg	_sSERIAL_CFG		Reception completion setting
	StartTrig	_eSERIAL_START	_SERIAL_START_-NONE	
	StartCode	ARRAY[0..1] OF BYTE	[2(16#0)]	
	EndTrig	_eSERIAL_END	_SERI-AL_END_CODE1	
	EndCode	ARRAY[0..1] OF BYTE	[16#0D,16#00]	End code: CR
	RcvSizeCfg	UINT	0	
	Option	_sSERIAL_RCV_OPTION		
	TimeOut	TIME	TIME#0s	
	LastDatRcv	BOOL	FALSE	
	ClearBuf	BOOL	FALSE	

External Variables	Variable	Data type	Comment
	E001_NX_Unit_I_O_Data_Active_Status_63	ARRAY[0..63] OF BOOL	• Usage of I/O data for 63 NX Units. • If the relevant Unit number is 1, E001_NX-_Unit_I_O_Data_Active_Status_63[1] is used.
	N1_Node_location_information	_sNXUNIT_ID	Device variable to specify NX-CIF210

```
// Condition setting
RS_instance1(Set:=Trigger AND E001_NX_Unit_I_O_Data_Active_Status_63[1]
             Reset1:=OperatingEnd,
             Q1=>Operating);
R_Trigger_instance(Clk:=Operating);
IF ( (R_Trigger_instance.Q=TRUE) ) THEN
           DevicePort.DeviceType:=_eDEVICE_TYPE#_DeviceNXUnit;
           DevicePort.NxUnit:=N1_Node_location_information;
           DevicePort.PortNo:=2;
           IF( (SelectSendBuf=TRUE) THEN
                     IF(SelectRcvBuf=TRUE) THEN
                              BufKind:=_eSERIAL_BUF_KIND#_BUF_SENDRCV;
                     ELSE
                              BufKind:=_eSERIAL_BUF_KIND#_BUF_SEND;
                     END_IF;
           ELSE
                     IF (SelectRcvBuf=TRUE) THEN
                               BufKind:=_eSERIAL_BUF_KIND#_BUF_RCV;
                     ELSE
                               BufKind:=_eSERIAL_BUF_KIND#_BUF_SENDRCV;
                     END_IF
           END_IF;
END_IF;

// Execute buffer clear
NX_SerialBufClear_instance(Execute:=Operating,
                           DevicePort:=DevicePort,
                           BufKind:=BufKind);

//
RS_instane2(Set:=NX_SerialBufClear.Done AND
E001_NX_Unit_I_O_Data_Active_Status_63[1],
            Reset1:=NX_SerialRcv_instance.Done OR NX_SerialRcv_instance.Error,
            Q1=>Rcving);

//
NX_SerialRcv_instance(Execute:=Rcving,
                      DevicePort:=DevicePort,
                      RcvDat:=RcvDat[0],
                      Size:=Size,
                      RcvCfg:=RcvCfg,
                      Option:=Option);
```

NX_SerialStartMon

The NX_SerialStartMon instruction starts serial line monitoring of an NX-series Communications Interface Unit.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
NX_SerialStart-Mon	Start Serial Line Monitor-ing	FB		NX_SerialStartMon_instance(Execute, DevicePort, Continuous, TimeOut, Done, Busy, Error, ErrorID);

Precautions for Correct Use

You cannot use this instruction for an Option Board for the NX1P2 CPU Unit.

Version Information

A CPU Unit with unit version 1.11 or later and Sysmac Studio version 1.15 or higher are required to use this instruction.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
DevicePort	Device port	Input	Object that represents a device port	---	---	---
Continuous	Continu-ous moni-toring		Serial line monitor opera-tion method TRUE: Continuous FALSE: One-shot	Depends on data type.	---	FALSE
TimeOut	Timeout time		2.0 s when the timeout time is set to 0	Depends on data type.	0.1 s	0

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
DevicePort	Refer to *Function* for details on the structure _sDEVICE_PORT.																			
Continuous	OK																			
TimeOut							OK													

Function

The NX_SerialStartMon instruction starts serial line monitoring of an NX-series Communications Interface Unit.

This instruction ends normally after serial line monitoring starts.

The data type of the *DevicePort* input variable is structure _sDEVICE_PORT. The specifications are as follows:

Variables	Meaning	Description	Data type	Valid range	Unit	Default
DevicePort	Device port	Object that represents a device port	_sDEVICE_PORT	---	---	---
DeviceType	Device type	Type of the device to specify	_eDEVICE_TYPE	_DeviceNXUnit _DeviceEcatSlave _DeviceOptionBoard	---	---
NxUnit	Specified Unit	NX Unit to control	_sNXUNIT_ID	---	---	---
EcatSlave	Specified slave	EtherCAT slave to control	_sECAT_ID	---	---	---
OptBoard	Specified Option Board	Option Board to control	_sOPTBOARD_ID	---	---	---
Reserved	Reserved	Reserved	Reserved	---	---	---
PortNo	Port number	Port number 1: Port 1 2: Port 2	USINT	Depends on data type.	---	---

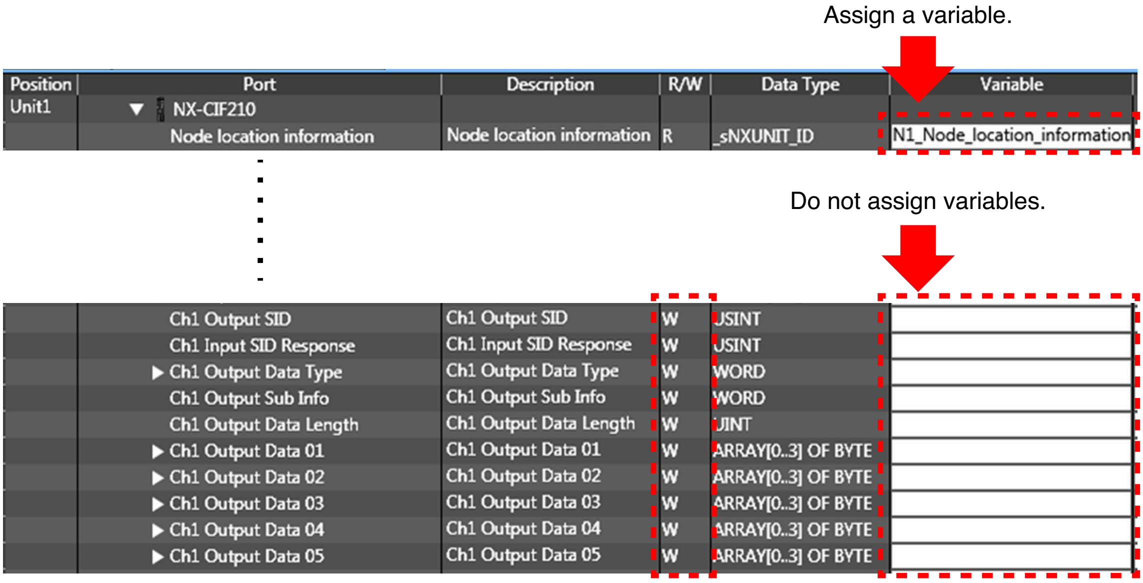
Use *DeviceType* to specify the device type. Set this to _DeviceNXUnit for an NX Unit. The variable used to specify the device is determined by the specified device type.

In this instruction, *NxUnit* is used to specify the device. *EcatSlave* and *OptBoard* are not used.

To *NxUnit*, pass the device variable that is assigned to the node location information on the I/O Map for the device to specify.

If you use this instruction, be sure to assign a device variable to the node location information. Do not assign device variables to any I/O ports following the node location information that are indicated by “W” under the R/W column.

The figure below is an example of using this instruction for port 1 on an NX-CIF210.



Refer to the *Sysmac Studio Version 1 Operation Manual* (Cat. No. W504-E1-07 or later) for details on assigning a device variable to the node location information.

Use *PortNo* to specify the port number.

1: Port 1

2: Port 2

The data type of *DeviceType* is enumerated type _eDEVICE_TYPE.

The meanings of the enumerators of enumerated type _eDEVICE_TYPE are as follows:

Enumerator	Meaning
_DeviceNXUnit	NX Unit is specified.
_DeviceEcatSlave	EtherCAT slave is specified.
_DeviceOptionBoard	Option Board is specified.

In this instruction, you can specify _DeviceNXUnit.

When the *Continuous* input variable is TRUE, continuous monitoring is selected and the monitoring is continued until the NX_SerialStopMon instruction is executed.

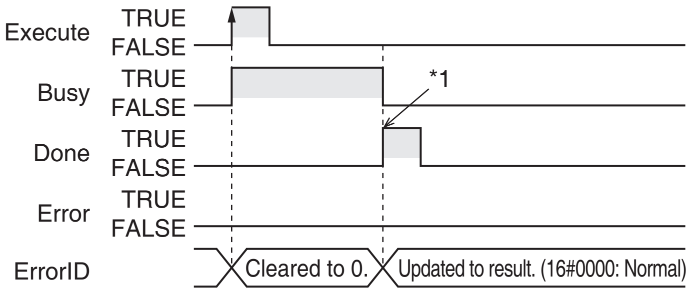
When the *Continuous* input variable is FALSE, one-shot monitoring is selected and serial line monitoring is continued until the buffer becomes full or the NX_SerialStopMon instruction is executed.

An error occurs if this instruction is executed for Units other than NX-series Communications Interface Units.

Timing Charts

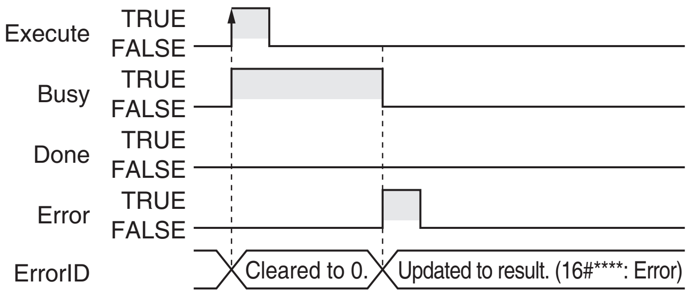
The following figures show the timing charts.

● Normal end



*1 Serial line monitoring is started.

● Error end



Related System-defined Variables

Name	Meaning	Data type	Description
_NXB_UnitIOActiveTbl	NX Unit I/O Data Active Status	ARRAY[0..8] OF BOOL	• This status tells the NX Units whether I/O data communications can be processed. • The subscript of the array corresponds to the NX Unit numbers. A subscript of 0 means the NX bus master.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- A compiling error will occur if you use this instruction in an event task. Do not use this instruction in event tasks.
- “CIF Unit Initialized” may occur when the NX-series Communications Interface Unit is restarted. Send or receive the data again, if necessary.
- If you use this instruction, do not assign device variables to any I/O ports that are indicated by “W” under the R/W column on the I/O Map Tab Page in the Sysmac Studio for the applicable NX-series Communications Interface Unit.
- An error will occur in the following cases. *Error* will change to TRUE.
  - A value that is out of range was set for *DevicePort.DevicePortType* or *DevicePort.PortNo*.
  - The Unit, Option Board, or port specified with *DevicePort* does not exist.
  - If more than 32 instructions from the NX_SerialSend instruction, NX_SerialRcv instruction, NX_-ModbusRtuCmd instruction, NX_ModbusRtuRead instruction, NX_ModbusRtuWrite instruction, NX_SerialSigCtl instruction, NX_SerialSigRead instruction, NX_SerialStatusRead instruction, NX_SerialBufClear instruction, NX_SerialStartMon instruction and NX_SerialStopMon instruction are executed at the same time.
  - This instruction is executed with a device port variable that is the same as the one specified for the instruction which is still being executed. In this case, the instruction which is still being executed is one of the followings.
    NX_SerialSigCtl instruction, NX_SerialSigRead instruction, NX_SerialStatusRead instruction, NX_SerialBufClear instruction, NX_SerialStartMon instruction, and NX_SerialStopMon instruction.
  - Timeout time elapsed during serial communications.
  - This instruction is executed for Units other than NX-series Communications Interface Units.

NX_SerialStopMon

The NX_SerialStopMon instruction stops serial line monitoring of an NX-series Communications Interface Unit.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
NX_SerialStop-Mon	Stop Serial Line Monitor-ing	FB	NX_SerialStopMon_instance NX_SerialStopMon Execute — Done DevicePort — Busy TimeOut — Error ErrorID	NX_SerialStopMon_instance(Execute, DevicePort, TimeOut, Done, Busy, Error, ErrorID);

Precautions for Correct Use

You cannot use this instruction for an Option Board for the NX1P2 CPU Unit.

Version Information

A CPU Unit with unit version 1.11 or later and Sysmac Studio version 1.15 or higher are required to use this instruction.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
DevicePort	Device port	Input	Object that represents a device port	---	---	---
TimeOut	Timeout time		2.0 s when the timeout time is set to 0	Depends on data type.	0.1 s	0

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
DevicePort	Refer to *Function* for details on the structure _sDEVICE_PORT.																			
TimeOut							OK													

Function

The NX_SerialStopMon instruction stops serial line monitoring of an NX-series Communications Interface Unit.

This instruction ends normally after serial line monitoring stops.

The data type of the *DevicePort* input variable is structure _sDEVICE_PORT. The specifications are as follows:

Variables		Meaning	Description	Data type	Valid range	Unit	Default
DevicePort		Device port	Object that represents a device port	_sDEVICE_PORT	---	---	---
	DeviceType	Device type	Type of the device to specify	_eDEVICE_TYPE	_DeviceNXUnit _DeviceEcatSlave _DeviceOptionBoard	---	---
	NxUnit	Specified Unit	NX Unit to control	_sNXUNIT_ID	---	---	---
	EcatSlave	Specified slave	EtherCAT slave to control	_sECAT_ID	---	---	---
	OptBoard	Specified Option Board	Option Board to control	_sOPTBOARD_ID	---	---	---
	Reserved	Reserved	Reserved	Reserved	---	---	---
	PortNo	Port number	Port number 1: Port 1 2: Port 2	USINT	Depends on data type.	---	---

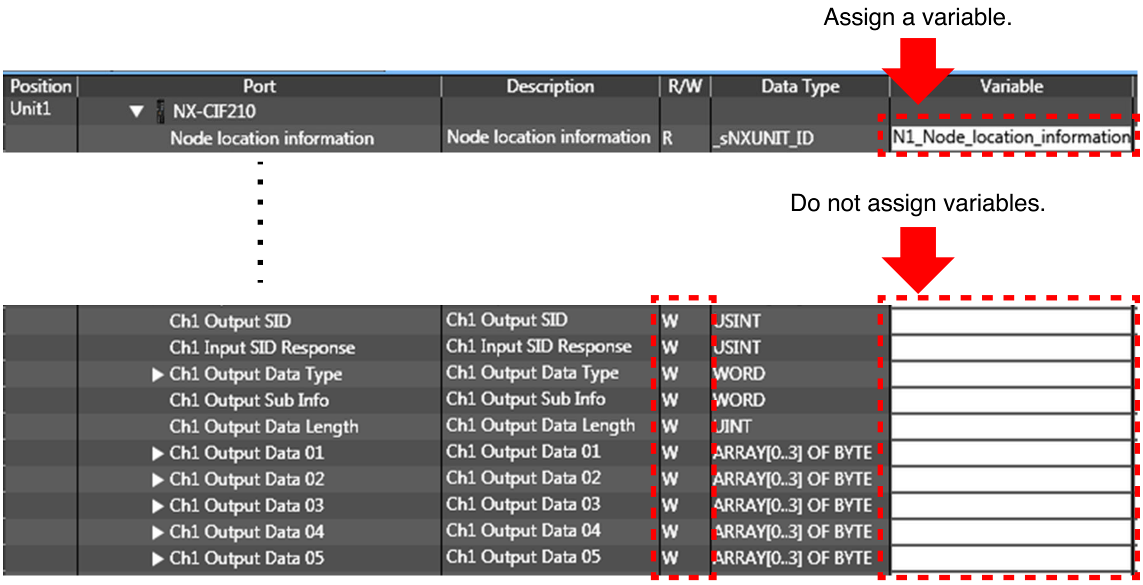
Use *DeviceType* to specify the device type. Set this to _DeviceNXUnit for an NX Unit. The variable used to specify the device is determined by the specified device type.

In this instruction, *NxUnit* is used to specify the device. *EcatSlave* and *OptBoard* are not used.

To *NxUnit*, pass the device variable that is assigned to the node location information on the I/O Map for the device to specify.

If you use this instruction, be sure to assign a device variable to the node location information. Do not assign device variables to any I/O ports following the node location information that are indicated by “W” under the R/W column.

The figure below is an example of using this instruction for port 1 on an NX-CIF210.



Refer to the *Sysmac Studio Version 1 Operation Manual* (Cat. No. W504-E1-07 or later) for details on assigning a device variable to the node location information.

Use *PortNo* to specify the port number.

1: Port 1
2: Port 2

The data type of *DeviceType* is enumerated type _eDEVICE_TYPE.
The meanings of the enumerators of enumerated type _eDEVICE_TYPE are as follows:

Enumerator	Meaning
_DeviceNXUnit	NX Unit is specified.
_DeviceEcatSlave	EtherCAT slave is specified.
_DeviceOptionBoard	Option Board is specified.

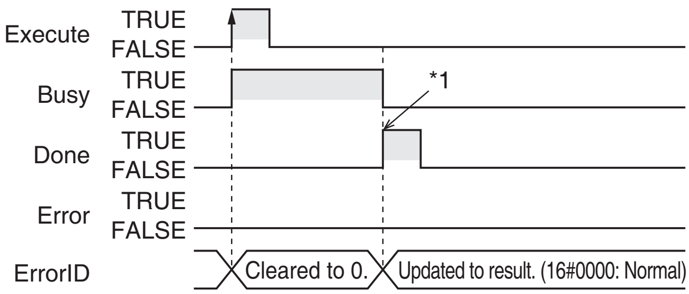
In this instruction, you can specify _DeviceNXUnit.

An error occurs if this instruction is executed for Units other than NX-series Communications Interface Units.

Timing Charts

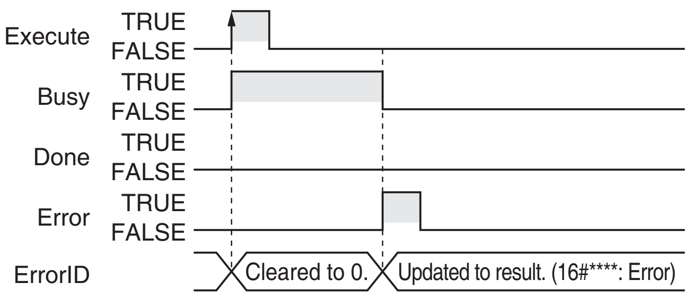
The following figures show the timing charts.

● **Normal end**



*1 Serial line monitoring is stopped.

● **Error end**



Related System-defined Variables

Name	Meaning	Data type	Description
_NXB_UnitIOActiveTbl	NX Unit I/O Data Active Status	ARRAY[0..8] OF BOOL	• This status tells the NX Units whether I/O data communications can be processed. • The subscript of the array corresponds to the NX Unit numbers. A subscript of 0 means the NX bus master.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- A compiling error will occur if you use this instruction in an event task. Do not use this instruction in event tasks.
- “CIF Unit Initialized” may occur when the NX-series Communications Interface Unit is restarted. Send or receive the data again, if necessary.
- If you use this instruction, do not assign device variables to any I/O ports that are indicated by “W” under the R/W column on the I/O Map Tab Page in the Sysmac Studio for the applicable NX-series Communications Interface Unit.
- An error will occur in the following cases. *Error* will change to TRUE.
  - A value that is out of range was set for *DevicePort.DevicePortType* or *DevicePort.PortNo*.
  - The Unit, Option Board, or port specified with *DevicePort* does not exist.
  - If more than 32 instructions from the NX_SerialSend instruction, NX_SerialRcv instruction, NX_ModbusRtuCmd instruction, NX_ModbusRtuRead instruction, NX_ModbusRtuWrite instruction, NX_SerialSigCtl instruction, NX_SerialSigRead instruction, NX_SerialStatusRead instruction, NX_SerialBufClear instruction, NX_SerialStartMon instruction and NX_SerialStopMon instruction are executed at the same time.
  - This instruction is executed with a device port variable that is the same as the one specified for the instruction which is still being executed. In this case, the instruction which is still being executed is one of the followings.
    NX_SerialSigCtl instruction, NX_SerialSigRead instruction, NX_SerialStatusRead instruction, NX_SerialBufClear instruction, NX_SerialStartMon instruction, and NX_SerialStopMon instruction.
  - Timeout time elapsed during serial communications.
  - This instruction is executed for Units other than NX-series Communications Interface Units.

SD Memory Card Instructions

FileWriteVar

The FileWriteVar instruction writes the value of a variable to the specified file in the SD Memory Card. The value is written in binary format.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FileWriteVar	Write Variable to File	FB	FileWriteVar_instance FileWriteVar Execute Done FileName Busy WriteVar Error OverWrite ErrorID	FileWriteVar_instance(Execute, FileName, WriteVar, OverWrite, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
FileName	File name	Input	Name of file to which to write variable	66 bytes max. (65 single-byte alphanumeric characters plus the final NULL character)	---	''
WriteVar	Variable		Variable to write	Depends on data type.		*
OverWrite	Overwrite enable		TRUE: Enable overwrite. FALSE: Prohibit overwrite.			FALSE

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileName																				OK
WriteVar	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	An enumeration, array, array element, structure, or structure member can also be specified.																			
OverWrite	OK																			

Function

The FileWriteVar instruction writes the value of variable *WriteVar* to the file specified by *FileName* in the SD Memory Card. The value is written in binary format. You can specify an enumeration, array, array element, structure, or structure member for *WriteVar*.

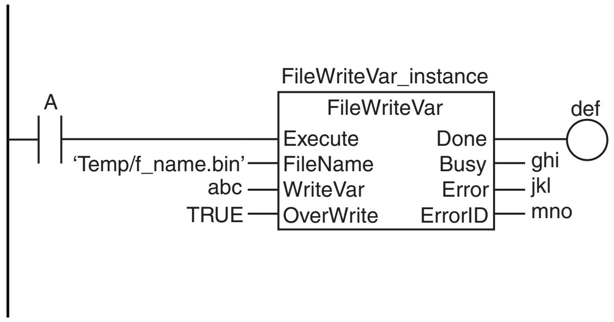
If a file with the name *FileName* does not exist on the SD Memory Card, it is created. *FileName* includes the path. If a specified directory does not exist in the SD Memory Card, it is created. However, the directory is created only when only the lowest directory level of the specified path does not exist.

If a file with the name *FileName* already exists in the SD Memory Card, the following processing is performed depending on the value of overwrite enable *OverWrite*.

Value of *OverWrite*	Processing
TRUE (Enable overwrite.)	The existing file is overwritten.
FALSE (Prohibit overwrite.)	The file is not overwritten and an error occurs.

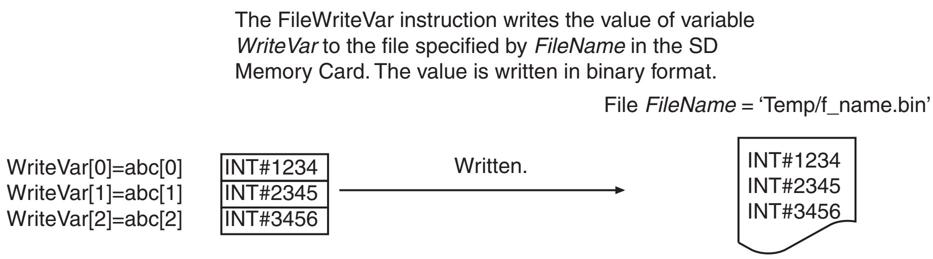
The following figure shows a programming example. The contents of array variable abc[0] is written to a file named 'Temp/f_name.bin.' Variable abc is an INT array variable with three elements.

LD



ST

```
FileWriteVar_instance(A, 'Temp/f_name.bin', abc,
    TRUE, def, ghi, jkl, mno);
```



Related System-defined Variables

Name	Meaning	Data type	Description
_Card1Ready	SD Memory Card Ready Flag	BOOL	This flag indicates if the SD Memory Card can be accessed by instructions and communications commands.*1 TRUE: Can be used. FALSE: Cannot be used.
_Card1Protect*2	SD Memory Card Write Protected Flag	BOOL	This flag indicates if the SD Memory Card is write protected when it is inserted and ready to use. TRUE: Write protected. FALSE: Not write protected.
_Card1Err*2	SD Memory Card Error Flag	BOOL	This flag indicates if an unspecified SD Memory Card (e.g., an SDHC card) is mounted or if the format is incorrect (i.e., not FAT16 or corrupted). TRUE: Error. FALSE: No error.
_Card1Access*2	SD Memory Card Access Flag	BOOL	This flag indicates if the SD Memory Card is currently being accessed. TRUE: Being accessed. FALSE: Not being accessed.
_Card1PowerFail	SD Memory Card Power Interruption Flag	BOOL	This flag indicates if an error occurred in completing processing when power was interrupted during access*3. This flag is not cleared automatically. TRUE: Error. FALSE: No error.

*1 For the NJ/NX-series, it is a precondition that the SD Memory Card is physically inserted and mounted normally. For an NY-series Controller, it is a precondition that the shared folder is detected by the Controller.

*2 These variables are not used for the NY-series Controller. They are fixed to FALSE.

*3 For the NJ/NX-series, this indicates an access to the SD Memory Card. For an NY-series Controller, this indicates an access to the shared folder (Virtual SD Memory Card).

Additional Information

The root directory of the file name is the top level of the SD Memory Card.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- Always use a variable for the input parameter to pass to *WriteVar*. A building error will occur if a constant is passed.
- If *WriteVar* is an enumeration, you cannot directly pass an enumerator to it. A building error will occur if an enumerator is passed to it directly.
- If the specified file is larger than the size of *WriteVar*, an error does not occur and only data that corresponds to the size of *WriteVar* is written. Once this instruction is executed, the specified file is reduced to the size of *WriteVar*.
- Data is written in byte increments. The lower bytes are written before the upper bytes (little endian).
- If *WriteVar* is a structure, adjustment areas between members may be inserted depending on the composition.
- Do not simultaneously access the same file. Perform exclusive control of SD Memory Card instructions in the user program.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The SD Memory Card is not in a usable condition.
  - The SD Memory Card is write protected.
  - There is insufficient space available on the SD Memory Card.
  - The value of *FileName* is not a valid file name.
  - The maximum number of files or directories is exceeded.
  - A file with the name *FileName* already exits and the file is being accessed.
  - A file with the name *FileName* already exits and the value of *OverWrite* is FALSE.
  - A file with the name *FileName* already exits and the file is write protected.
  - If more than four SD Memory Card instructions that do not have a *FileID* variable (i.e., FileWriteVar, FileReadVar, FileCopy, DirCreate, FileRemove, DirRemove, and FileRename) are executed at the same time.
  - The value of *FileName* exceeds the maximum number of bytes allowed in a file name.
  - An error that prevents access occurs during SD Memory Card access.

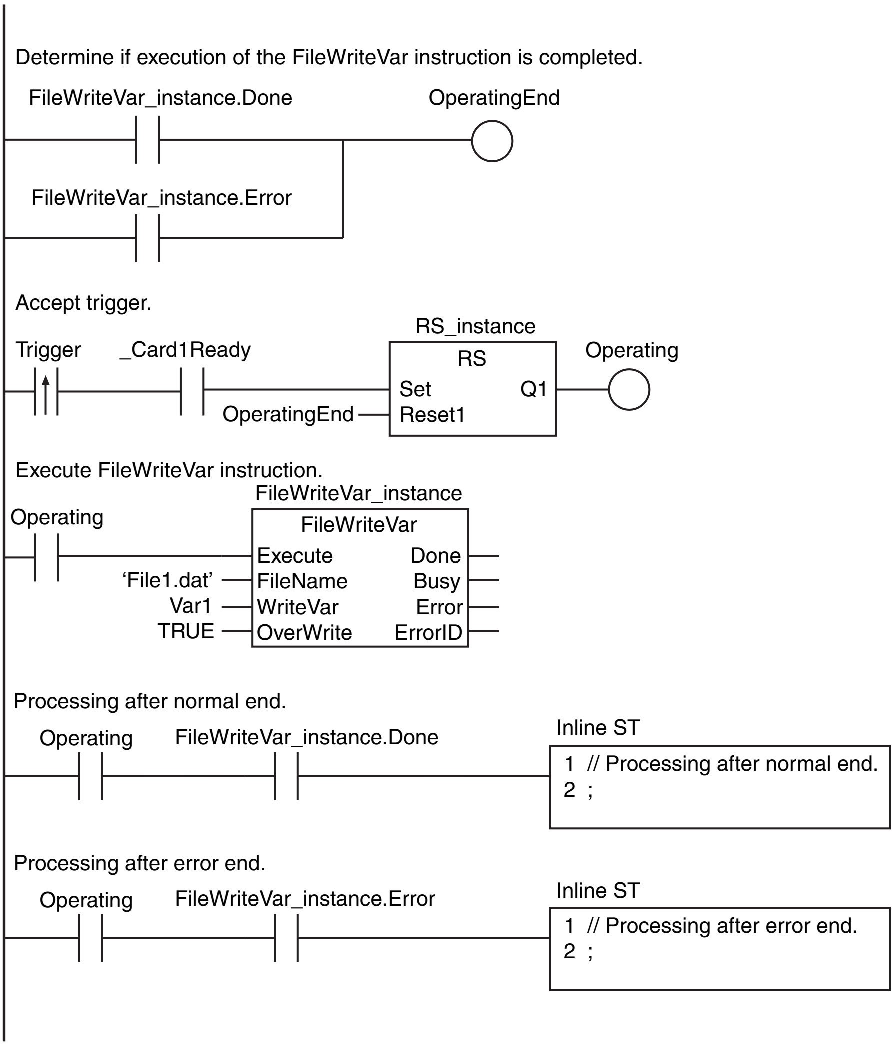
Sample Programming

This sample writes all of array variable *Var1[]* to the file 'File1.dat.'

LD

Internal Variables	Variable	Data type	Initial value	Comment
	OperatingEnd	BOOL	FALSE	Processing completed.
	Trigger	BOOL	FALSE	Execution condition
	Operating	BOOL	FALSE	Processing
	Var1	ARRAY[0..999] OF INT	[1000(0)]	Write data
	RS_instance	RS		
	FileWriteVar_instance	FileWriteVar		

External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag



ST

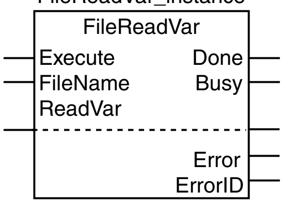
Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	FALSE	Execution condition
	LastTrigger	BOOL	FALSE	Value of *Trigger* from previous task period
	OperatingStart	BOOL	FALSE	Processing started.
	Operating	BOOL	FALSE	Processing
	Var1	ARRAY[0..999] OF INT	[1000(0)]	Variable
	FileWriteVar_instance	FileWriteVar		

External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag

```
// Detect when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Card1Ready=TRUE) ) THEN
    OperatingStart:=TRUE;
    Operating     :=TRUE;
END_IF;
LastTrigger:=Trigger;

// Initialize FileWriteVar instruction.
IF (OperatingStart=TRUE) THEN
    FileWriteVar_instance(
        Execute    :=FALSE,
        WriteVar   :=Var1);
    OperatingStart  :=FALSE;
END_IF;

// Execute FileWriteVar instruction.
IF (Operating=TRUE) THEN
    FileWriteVar_instance(
        Execute  :=TRUE,
        FileName :='File1.dat', // File name
        WriteVar :=Var1,        // Variable
        OverWrite:=TRUE);       // Enable overwrite.

    IF (FileWriteVar_instance.Done=TRUE) THEN
        // Processing after normal end.
        Operating:=FALSE;
    END_IF;

    IF (FileWriteVar_instance.Error=TRUE) THEN
        // Processing after error end.
        Operating:=FALSE;
    END_IF;
END_IF;
```

FileReadVar

The FileReadVar instruction reads the contents of the specified file on the SD Memory Card as binary data and writes it to a variable.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FileReadVar	Read Variable from File	FB		FileReadVar_instance(Execute, FileName, ReadVar, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
FileName	File name	Input	Name of file to read	66 bytes max. (65 single-byte alphanumeric characters plus the final NULL character)	---	''
ReadVar	Variable to write	In-out	Variable to which to write the value that was read	Depends on data type.	---	---

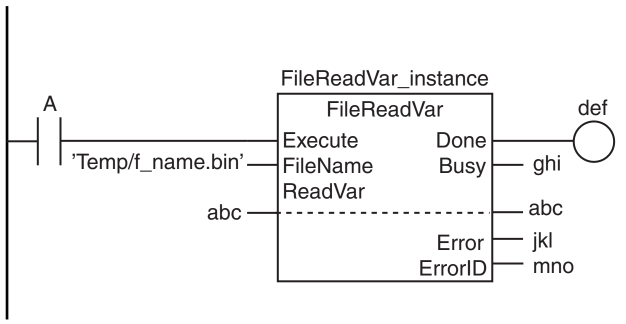
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileName																				OK
ReadVar	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	An enumeration, array, array element, structure, or structure member can also be specified.																			

Function

The FileReadVar instruction reads the contents of the file specified by *FileName* from the SD Memory Card as binary data. The contents that is read is assigned to variable to write *ReadVar*. You can specify an enumeration, array, array element, structure, or structure member for *ReadVar*.

The following figure shows a programming example. Here, the contents of the file called ‘Temp/f_name.bin’ is read and written to the array variable abc[]. Variable *abc* is an INT array variable with three elements.

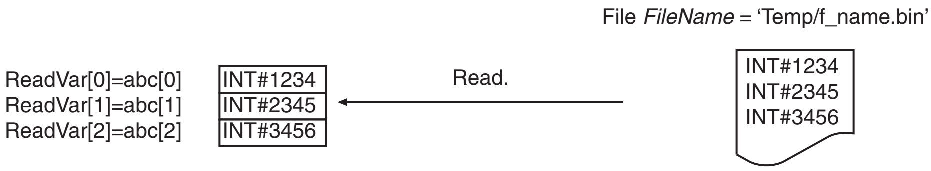
LD



ST

```
FileReadVar_instance(A, ‘Temp/f_name.bin’, abc,
                     def, ghi, jkl, mno);
```

The FileReadVar instruction reads the contents of the file specified by *FileName* from the SD Memory Card as binary data and assigns it to variable *ReadVar*.



Related System-defined Variables

Name	Meaning	Data type	Description
_Card1Ready	SD Memory Card Ready Flag	BOOL	This flag indicates if the SD Memory Card can be accessed by instructions and communications commands.*1 TRUE: Can be used. FALSE: Cannot be used.
_Card1Protect*2	SD Memory Card Write Protected Flag	BOOL	This flag indicates if the SD Memory Card is write protected when it is inserted and ready to use. TRUE: Write protected. FALSE: Not write protected.
_Card1Err*2	SD Memory Card Error Flag	BOOL	This flag indicates if an unspecified SD Memory Card (e.g., an SDHC card) is mounted or if the format is incorrect (i.e., not FAT16 or corrupted). TRUE: Error. FALSE: No error.
_Card1Access*2	SD Memory Card Access Flag	BOOL	This flag indicates if the SD Memory Card is currently being accessed. TRUE: Being accessed. FALSE: Not being accessed.
_Card1PowerFail	SD Memory Card Power Interruption Flag	BOOL	This flag indicates if an error occurred in completing processing when power was interrupted during access*3. This flag is not cleared automatically. TRUE: Error. FALSE: No error.

*1 For the NJ/NX-series, it is a precondition that the SD Memory Card is physically inserted and mounted normally. For an NY-series Controller, it is a precondition that the shared folder is detected by the Controller.

*2 These variables are not used for the NY-series Controller. They are fixed to FALSE.

*3 For the NJ/NX-series, this indicates an access to the SD Memory Card. For an NY-series Controller, this indicates an access to the shared folder.

Additional Information

The root directory of the file name is the top level of the SD Memory Card.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- If the specified file is larger than the size of *ReadVar*, an error does not occur and only data that corresponds to the size of *ReadVar* is read.
- If the specified file is smaller than the size of *ReadVar*, an error does not occur and only data that corresponds to the size of the specified file is read. The remaining area in *ReadVar* will retain the values from before execution of this instruction.
- Data is read in byte increments. The lower bytes are read before the upper bytes (little endian).
- If *ReadVar* is a structure, adjustment areas between members may be inserted depending on the composition.
- Do not simultaneously access the same file. Perform exclusive control of SD Memory Card instructions in the user program.
- You cannot specify a device variable for *ReadVar*. If you specify a device variable, the value that was read is not assigned to *ReadVar*.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The SD Memory Card is not in a usable condition.
  - The file specified by *FileName* does not exist.
  - The value of *FileName* is not a valid file name.
  - The file specified by *FileName* is being accessed.
  - If more than four SD Memory Card instructions that do not have a *FileID* variable (i.e., FileWriteVar, FileReadVar, FileCopy, DirCreate, FileRemove, DirRemove, and FileRename) are executed at the same time.
  - The value of *FileName* exceeds the maximum number of bytes allowed in a file name.
  - An error that prevents access occurs during SD Memory Card access.

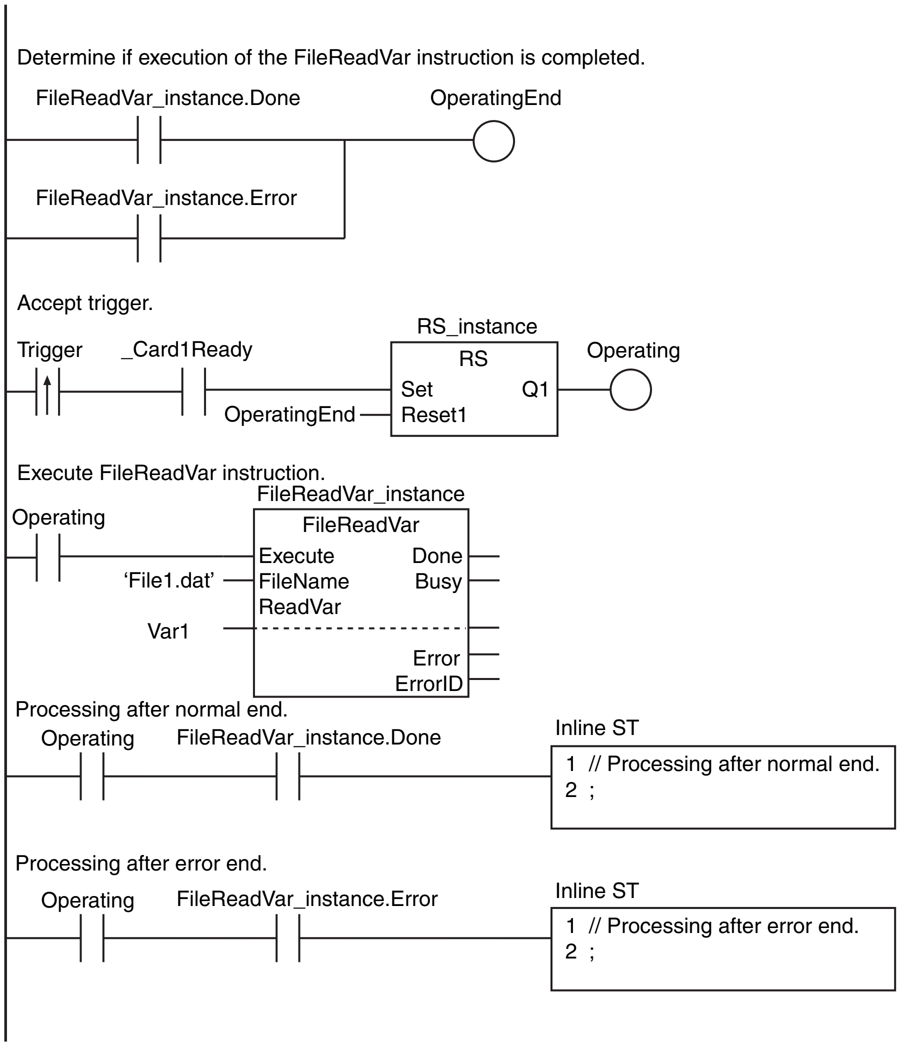
Sample Programming

This sample reads the contents of the file 'File1.dat' and stores it in array variable *Var1*.

LD

Internal Variables	Variable	Data type	Initial value	Comment
	OperatingEnd	BOOL	FALSE	Processing completed.
	Trigger	BOOL	FALSE	Execution condition
	Operating	BOOL	FALSE	Processing
	Var1	ARRAY[0..999] OF INT	[1000(0)]	Read data
	RS_instance	RS		
	FileReadVar_instance	FileReadVar		

External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag


Determine if execution of the FileReadVar instruction is completed.
FileReadVar_instance.Done
OperatingEnd
FileReadVar_instance.Error
Accept trigger.
RS_instance
Trigger
_Card1Ready
RS
Operating
Set
Q1
OperatingEnd
Reset1
Execute FileReadVar instruction.
FileReadVar_instance
Operating
FileReadVar
Execute
Done
'File1.dat'
FileName
Busy
ReadVar
Var1
Error
ErrorID
Processing after normal end.
Operating
FileReadVar_instance.Done
Inline ST
1 // Processing after normal end.
2 ;
Processing after error end.
Operating
FileReadVar_instance.Error
Inline ST
1 // Processing after error end.
2 ;

ST

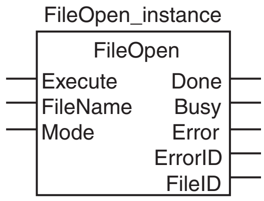
Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	FALSE	Execution condition
	LastTrigger	BOOL	FALSE	Value of *Trigger* from previous task period
	OperatingStart	BOOL	FALSE	Processing started.
	Operating	BOOL	FALSE	Processing
	Var1	ARRAY[0..999] OF INT	[1000(0)]	Variable to read
	FileReadVar_instance	FileReadVar		

External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag

```
// Detect when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Card1Ready=TRUE) ) THEN
    OperatingStart:=TRUE;
    Operating     :=TRUE;
END_IF;
LastTrigger:=Trigger;

// Initialize FileReadVar instruction.
IF (OperatingStart=TRUE) THEN
    FileReadVar_instance(
        Execute  :=FALSE,
        ReadVar  :=Var1);
    OperatingStart:=FALSE;
END_IF;

// Execute FileReadVar instruction.
IF (Operating=TRUE) THEN
    FileReadVar_instance(
        Execute  :=TRUE,
        FileName :='File1.dat', // File name
        ReadVar  :=Var1);       // Variable to read

    IF (FileReadVar_instance.Done=TRUE) THEN
        // Processing after normal end.
        Operating:=FALSE;
    END_IF;

    IF (FileReadVar_instance.Error=TRUE) THEN
        // Processing after error end.
        Operating:=FALSE;
    END_IF;
END_IF;
```

FileOpen

The FileOpen instruction opens the specified file in the SD Memory Card.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FileOpen	Open File	FB		FileOpen_instance(Execute, FileName, Mode, Done, Busy, Error, ErrorID, FileID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
FileName	File name	Input	Name of file to open	66 bytes max. (65 single-byte alphanumeric characters plus the final NULL character)	---	''
Mode	Open mode	Input	Mode in which to open file	*	---	_READ_EXIST
FileID	File ID	Output	ID of file that was opened	Depends on data type.	---	---

* _READ_EXIST, _RDWR_EXIST, _WRITE_CREATE, _RDWR_CREATE, _WRITE_APPEND and _RDWR_APPEND

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileName																				OK
Mode	Refer to *Function* for the enumerators for the enumerated type _eFOPEN_MODE.																			
FileID				OK																

Function

The FileOpen instruction opens the file specified by *FileName* in the SD Memory Card in the mode specified by *Mode*. The result is output to file ID *FileID*. *FileID* is used to specify the file in other instructions, such as FileRead and FileWrite.

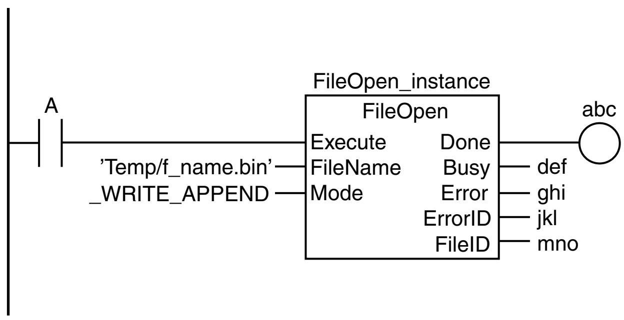
The data type of *Mode* is enumerated type _eFOPEN_MODE. The meanings of the enumerators are as follows:

Enumerator	Meaning
_READ_EXIST	Use this value to open a text file to read it. The file is read from the beginning.
_RDWR_EXIST	Use this value to open a file to read and write it. The file is read and written from the beginning.
_WRITE_CREATE	Use this value to open a file to write it. If the file already exists, the contents is discarded and the file size is set to 0. If the file does not exist, a new file is created. The file is written from the beginning. However, if the file already exists and it is write-protected, an error occurs and the file is not opened.

Enumerator	Meaning
_RDWR_CREATE	Use this value to open a file to read and write it. If the file already exists, the contents is discarded and the file size is set to 0. If the file does not exist, a new file is created. The file is read and written from the beginning.
_WRITE_APPEND	Use this value to open a file to append data to it. If the file does not exist, a new file is created. The data is appended to the end of the file. However, if the file already exists and it is write-protected, an error occurs and the file is not opened.
_RDWR_APPEND	Use this value to open a file to read and append data to it. If the file does not exist, a new file is created. The file is read from the beginning. The data is appended to the end of the file.

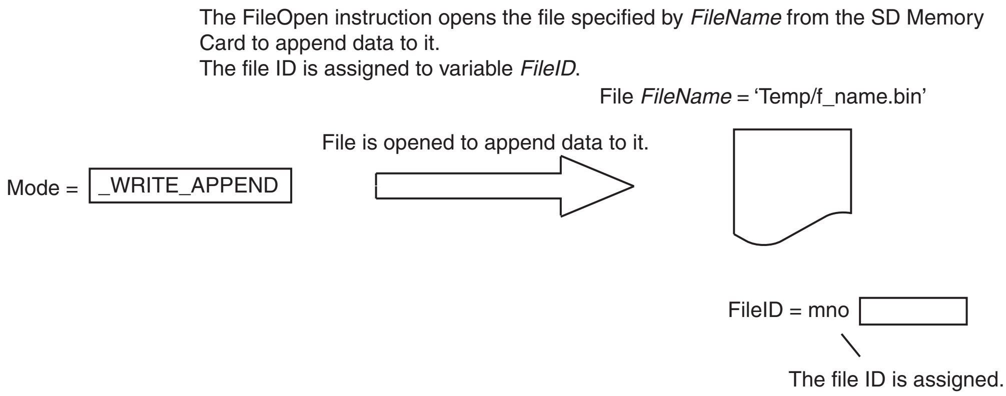
The following figure shows a programming example. The file named 'Temp/f_name.bin' is opened to append data to it. The file ID is assigned to variable mno.

LD



ST

```
FileOpen_instance(A, 'Temp/f_name.bin',
                  _WRITE_APPEND, abc,
                  def, ghi, jkl, mno);
```



Related System-defined Variables

Name	Meaning	Data type	Description
_Card1Ready	SD Memory Card Ready Flag	BOOL	This flag indicates if the SD Memory Card can be accessed by instructions and communications commands.*1 TRUE: Can be used. FALSE: Cannot be used.
_Card1Protect*2	SD Memory Card Write Protected Flag	BOOL	This flag indicates if the SD Memory Card is write protected when it is inserted and ready to use. TRUE: Write protected. FALSE: Not write protected.
_Card1Err*2	SD Memory Card Error Flag	BOOL	This flag indicates if an unspecified SD Memory Card (e.g., an SDHC card) is mounted or if the format is incorrect (i.e., not FAT16 or corrupted). TRUE: Error. FALSE: No error.
_Card1Access*2	SD Memory Card Access Flag	BOOL	This flag indicates if the SD Memory Card is currently being accessed. TRUE: Being accessed. FALSE: Not being accessed.
_Card1PowerFail	SD Memory Card Power Interruption Flag	BOOL	This flag indicates if an error occurred in completing processing when power was interrupted during access*3. This flag is not cleared automatically. TRUE: Error. FALSE: No error.

*1 For the NJ/NX-series, it is a precondition that the SD Memory Card is physically inserted and mounted normally. For an NY-series Controller, it is a precondition that the shared folder is detected by the Controller.

*2 These variables are not used for the NY-series Controller. They are fixed to FALSE.

*3 For the NJ/NX-series, this indicates an access to the SD Memory Card. For an NY-series Controller, this indicates an access to the shared folder.

Additional Information

The root directory of the file name is the top level of the SD Memory Card.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- This instruction must be executed before any of the following instructions: FileSeek, FileRead, FileWrite, FileGets, and FilePuts.
- You must use the FileClose instruction to close any file that is opened with this instruction after you finish using it.
- A value is stored in *FileID* when the instruction is completed. Specifically, it is stored when the value of *Done* changes from FALSE to TRUE.
- If a file is open when the operating mode of the CPU Unit is changed to PROGRAM mode or when a major fault level Controller error occurs, the file is closed by the system. Any read/write operations that are in progress are completed to the end.

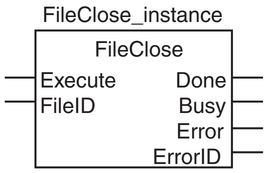
- For an NJ/NX-series CPU Unit, if a file is open when the power supply is stopped with the SD Memory Card power supply switch, the file is not corrupted. However, use the FileClose instruction to close the file, since the file remains open.
- For an NJ/NX-series CPU Unit, if a file is open and the SD Memory Card is removed before the SD Memory Card power supply switch is pressed, the contents of the file will sometimes be corrupted. Always turn OFF the power supply before removing the SD Memory Card.
- For an NJ/NX-series CPU Unit, if a file is open and the SD Memory Card is removed before the SD Memory Card power supply switch is pressed, the file will remain open. Use the FileClose instruction to close the file.
- For an NJ/NX-series CPU Unit, if a file is open when the power supply is stopped or the SD Memory Card is removed, the file will remain open, but it will not be possible to read or write the file even if the SD Memory Card is inserted again. To read/write the file, close the file and then open it again.
- Do not simultaneously access the same file. Perform exclusive control of SD Memory Card instructions in the user program.
- An error occurs in the following cases. *Error* will change to TRUE.
    - The SD Memory Card is not in a usable condition.
    - The SD Memory Card is write protected.
    - The value of *Mode* is _READ_EXIST or _RDWR_EXIST and the file specified with *FileName* does not exist.
    - The value of *FileName* is not a valid file name.
    - The maximum number of files or directories is exceeded.
    - The file specified by *FileName* is being accessed.
    - The file specified by *FileName* is write protected.
    - An attempt was made to open more than five files at the same time.
    - The value of *FileName* exceeds the maximum number of bytes allowed in a file name.
    - An error that prevents access occurs during SD Memory Card access.
    - The value of *Mode* is outside of the valid range.
    - For CPU Unit version 1.10 or later, if you try to open a file that is already open, a File Already in Use error occurs and the file ID of the open file is stored in the *FileID* output variable. The *FileID* output variable does not change if any other error occurs. For CPU Unit version 1.09 or earlier, 0 is stored in the *FileID* output variable if an error occurs.

Sample Programming

Refer to the sample programming for the following instructions: FileRead (page 2-1277), FileWrite (page 2-1285), FileGets (page 2-1293), and FilePuts (page 2-1301).

FileClose

The FileClose instruction closes the specified file in the SD Memory Card.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FileClose	Close File	FB		FileClose_instance(Execute, FileID, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
FileID	File ID	Input	ID of file to close	Depends on data type.	---	0

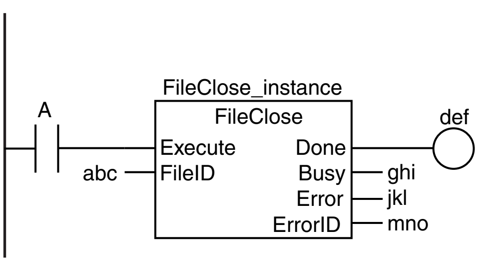
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileID				OK																

Function

The FileClose instruction closes the file specified by *FileID* in the SD Memory Card.

The following figure shows a programming example. Here, the file whose file ID is the value of variable *abc* is closed.

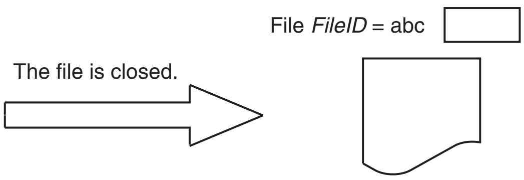
LD



ST

```
FileClose_instance(A, abc, def, ghi, jkl, mno);
```

The FileClose instruction closes the file specified by *FileID* in the SD Memory Card.



Related System-defined Variables

Name	Meaning	Data type	Description
_Card1Ready	SD Memory Card Ready Flag	BOOL	This flag indicates if the SD Memory Card can be accessed by instructions and communications commands.*1 TRUE: Can be used. FALSE: Cannot be used.
_Card1Protect*2	SD Memory Card Write Protected Flag	BOOL	This flag indicates if the SD Memory Card is write protected when it is inserted and ready to use. TRUE: Write protected. FALSE: Not write protected.
_Card1Err*2	SD Memory Card Error Flag	BOOL	This flag indicates if an unspecified SD Memory Card (e.g., an SDHC card) is mounted or if the format is incorrect (i.e., not FAT16 or corrupted). TRUE: Error. FALSE: No error.
_Card1Access*2	SD Memory Card Access Flag	BOOL	This flag indicates if the SD Memory Card is currently being accessed. TRUE: Being accessed. FALSE: Not being accessed.
_Card1PowerFail	SD Memory Card Power Interruption Flag	BOOL	This flag indicates if an error occurred in completing processing when power was interrupted during access*3. This flag is not cleared automatically. TRUE: Error. FALSE: No error.

*1 For the NJ/NX-series, it is a precondition that the SD Memory Card is physically inserted and mounted normally. For an NY-series Controller, it is a precondition that the shared folder is detected by the Controller.

*2 These variables are not used for the NY-series Controller. They are fixed to FALSE.

*3 For the NJ/NX-series, this indicates an access to the SD Memory Card. For an NY-series Controller, this indicates an access to the shared folder.

Additional Information

You must open files with the FileOpen instruction for the following instructions: FileSeek, FileRead, FileWrite, FileGets, and FilePuts.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- You must use the FileOpen instruction in advance to obtain the value for *FileID*.
- You must use this instruction to close any file that is opened with the FileOpen instruction after you finish using it.
- If a file is open when the operating mode of the CPU Unit is changed to PROGRAM mode or when a major fault level Controller error occurs, the file is closed by the system. Any read/write operations that are in progress are completed to the end.

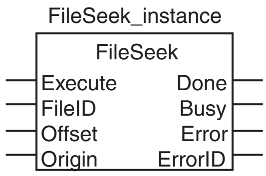
- For an NJ/NX-series CPU Unit, if a file is open when the power supply is stopped with the SD Memory Card power supply switch, the file is not corrupted. However, use the FileClose instruction to close the file, since the file remains open.
- For an NJ/NX-series CPU Unit, if a file is open and the SD Memory Card is removed before the SD Memory Card power supply switch is pressed, the contents of the file will sometimes be corrupted. Always turn OFF the power supply before removing the SD Memory Card.
- For an NJ/NX-series CPU Unit, if a file is open and the SD Memory Card is removed before the SD Memory Card power supply switch is pressed, the file will remain open. Use the FileClose instruction to close the file.
- For an NJ/NX-series CPU Unit, if a file is open when the power supply is stopped or the SD Memory Card is removed, the file will remain open, but it will not be possible to read or write the file even if the SD Memory Card is inserted again. To read/write the file, close the file and then open it again.
- Do not simultaneously access the same file. Perform exclusive control of SD Memory Card instructions in the user program.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The file specified by *FileID* does not exist.
  - The file specified by *FileID* is already closed.
  - The file specified by *FileID* is being accessed.
  - An error that prevents access occurs during SD Memory Card access.
  - The SD Memory Card is not in a usable condition.

Sample Programming

Refer to the sample programming for the following instructions: FileRead (page 2-1277), FileWrite (page 2-1285), FileGets (page 2-1293), and FilePuts (page 2-1301).

FileSeek

The FileSeek instruction sets a file position indicator in the specified file in the SD Memory Card.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FileSeek	Seek File	FB		FileSeek_instance(Execute, FileID, Offset, Origin, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
FileID	File ID	Input	ID of file in which to set file position indicator	Depends on data type.	---	0
Offset	Offset		Offset from *Origin*		Bytes	
Origin	Reference position		Reference position for file position indicator	_SEEK_SET, _SEEK_CUR, or _SEEK_END	---	_SEEK _SET

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileID				OK																
Offset												OK								
Origin	Refer to *Function* for the enumerators for the enumerated type _eFSEEK_ORIGIN.																			

Function

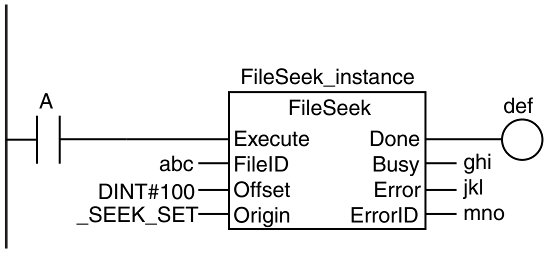
The FileSeek instruction sets a file position indicator in the file specified by file ID *FileID* in the SD Memory Card. A file position indicator is the position in a file at which to start reading or writing when an instruction such as the FileRead or FileWrite instruction is executed. For example, to read from the beginning of a file, set a file position indicator at the beginning of the file with the FileSeek instruction, and then execute the FileRead instruction. The file position indicator is set at offset *Offset* from reference position *Origin*.

The data type of *Origin* is enumerated type _eFSEEK_ORIGIN. The meanings of the enumerators are as follows:

Enumerator	Meaning
_SEEK_SET	Beginning of file
_SEEK_CUR	Location of current file position indicator
_SEEK_END	End of file

The following figure shows a programming example. A file position indicator is set at 100 bytes from the beginning of the file.

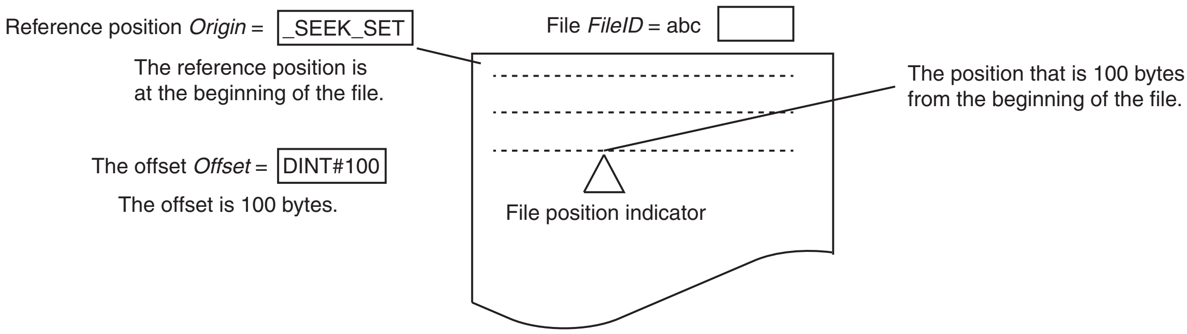
LD



ST

```
FileSeek_instance(A, abc, DINT#100,
                  _SEEK_SET, def,
                  ghi, jkl, mno);
```

The FileSeek instruction sets a file position indicator in the file specified by *FileID* in the SD Memory Card. The file position indicator is at the position that is *Offset* from the beginning of the file.



Related System-defined Variables

Name	Meaning	Data type	Description
_Card1Ready	SD Memory Card Ready Flag	BOOL	This flag indicates if the SD Memory Card can be accessed by instructions and communications commands.*1 TRUE: Can be used. FALSE: Cannot be used.
_Card1Protect*2	SD Memory Card Write Protected Flag	BOOL	This flag indicates if the SD Memory Card is write protected when it is inserted and ready to use. TRUE: Write protected. FALSE: Not write protected.
_Card1Err*2	SD Memory Card Error Flag	BOOL	This flag indicates if an unspecified SD Memory Card (e.g., an SDHC card) is mounted or if the format is incorrect (i.e., not FAT16 or corrupted). TRUE: Error. FALSE: No error.
_Card1Access*2	SD Memory Card Access Flag	BOOL	This flag indicates if the SD Memory Card is currently being accessed. TRUE: Being accessed. FALSE: Not being accessed.
_Card1PowerFail	SD Memory Card Power Interruption Flag	BOOL	This flag indicates if an error occurred in completing processing when power was interrupted during access*3. This flag is not cleared automatically. TRUE: Error. FALSE: No error.

*1 For the NJ/NX-series, it is a precondition that the SD Memory Card is physically inserted and mounted normally. For an NY-series Controller, it is a precondition that the shared folder is detected by the Controller.

*2 These variables are not used for the NY-series Controller. They are fixed to FALSE.

*3 For the NJ/NX-series, this indicates an access to the SD Memory Card. For an NY-series Controller, this indicates an access to the shared folder.

Precautions for Correct Use

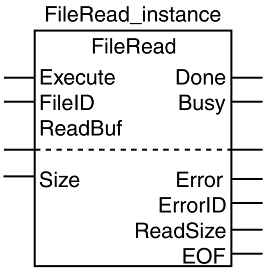
- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- You must use the FileOpen instruction to obtain the value for *FileID* before you execute this instruction.
- Do not simultaneously access the same file. Perform exclusive control of SD Memory Card instructions in the user program.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The value of *Origin* is outside of the valid range.
  - The SD Memory Card is not in a usable condition.
  - The file specified by *FileID* does not exist.
  - The file specified by *FileID* is being accessed.
  - The position specified by *Origin* and *Offset* exceeds the file size.
  - An error that prevents access occurs during SD Memory Card access.

Sample Programming

Refer to the sample programming for the following instructions: FileRead (page 2-1277) and FileWrite (page 2-1285).

FileRead

The FileRead instruction reads the data from the specified file in the SD Memory Card.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FileRead	Read File	FB		FileRead_instance(Execute, FileID, ReadBuf, Size, Done, Busy, Error, ErrorID, ReadSize, EOF);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
FileID	File ID	Input	ID of file to read	Depends on data type.	---	0
Size	Number of elements to read		Number of elements to read			1
ReadBuf[] (array)	Read buffer	In-out	Buffer in which to write data that was read	Depends on data type.	---	---
ReadSize	Number of read elements	Output	Number of elements that were actually read	Depends on data type.	---	---
EOF	End of file		Whether end of file was reached TRUE: Reached. FALSE: Not reached.			

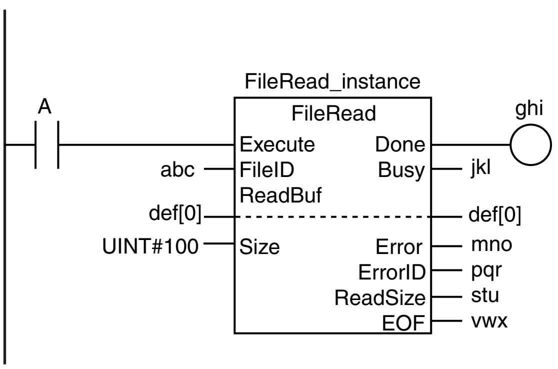
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileID				OK																
Size							OK													
ReadBuf[] (array)	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	Arrays enumerations or structures can also be specified.																			
ReadSize							OK													
EOF	OK																			

Function

The FileRead instruction reads the data from position of the file position indicator in the file specified by file ID *FileID* in the SD Memory Card. It then stores the data in read buffer *ReadBuf[]*. The file position indicator is set at the desired location in advance with the FileSeek instruction. The amount of data that is read is the size of the data type of *ReadBuf[]* times *Size*. You can specify an array of enumerations or structures for *ReadBuf[]*. The actual number of elements that were read is stored in *ReadSize*. Normally, *Size* and *ReadSize* will have the same values. If the amount of data from the file position indicator to the end of the file is smaller than *Size*, an error will not occur and the data to the end of the file is stored in *ReadBuf[]*. If that occurs, the value of *ReadSize* will be smaller than the value of *Size*. If data is read to the end of the file, end of file *EOF* changes to TRUE. Otherwise, the value of *EOF* will be FALSE.

The following figure shows a programming example. If the read buffer *def[]* is a BYTE array, 100 bytes of data is read from the file.

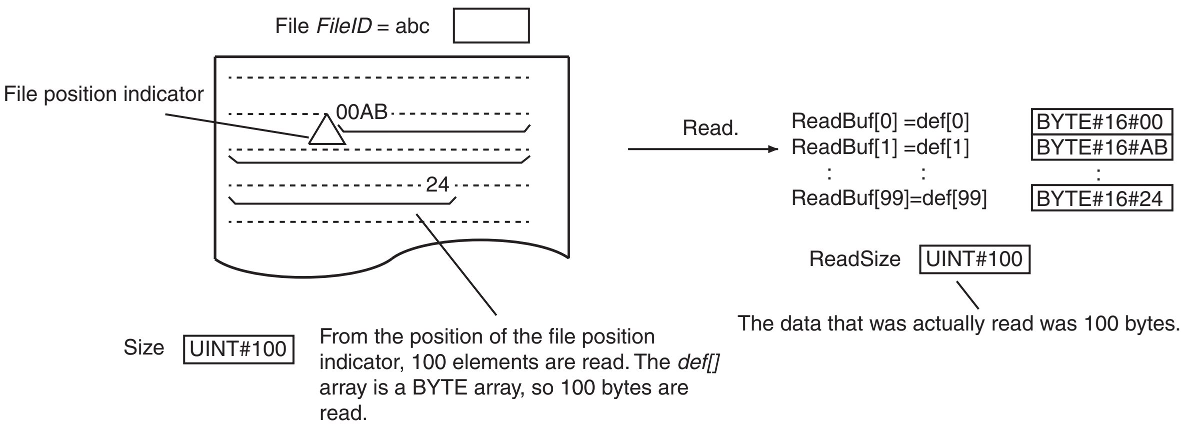
LD



ST

```
FileRead_instance(A, abc, def[0], UINT#100,
                  ghi, jkl, mno, pqr, stu, vwx);
```

The FileRead instruction reads *Size* elements from the position of the file position indicator in the file specified by *FileID* in the SD Memory Card. It then stores the data in read buffer *ReadBuf[]*. The actual data size that was read is output to *ReadSize*.



Related System-defined Variables

Name	Meaning	Data type	Description
_Card1Ready	SD Memory Card Ready Flag	BOOL	This flag indicates if the SD Memory Card can be accessed by instructions and communications commands.*1 TRUE: Can be used. FALSE: Cannot be used.

Name	Meaning	Data type	Description
_Card1Protect*2	SD Memory Card Write Protected Flag	BOOL	This flag indicates if the SD Memory Card is write protected when it is inserted and ready to use. TRUE: Write protected. FALSE: Not write protected.
_Card1Err*2	SD Memory Card Error Flag	BOOL	This flag indicates if an unspecified SD Memory Card (e.g., an SDHC card) is mounted or if the format is incorrect (i.e., not FAT16 or corrupted). TRUE: Error. FALSE: No error.
_Card1Access*2	SD Memory Card Access Flag	BOOL	This flag indicates if the SD Memory Card is currently being accessed. TRUE: Being accessed. FALSE: Not being accessed.
_Card1PowerFail	SD Memory Card Power Interruption Flag	BOOL	This flag indicates if an error occurred in completing processing when power was interrupted during access*3. This flag is not cleared automatically. TRUE: Error. FALSE: No error.

*1 For the NJ/NX-series, it is a precondition that the SD Memory Card is physically inserted and mounted normally. For an NY-series Controller, it is a precondition that the shared folder is detected by the Controller.

*2 These variables are not used for the NY-series Controller. They are fixed to FALSE.

*3 For the NJ/NX-series, this indicates an access to the SD Memory Card. For an NY-series Controller, this indicates an access to the shared folder.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- If the data is read to the end of the file and the size of the data is not evenly divisible by the size of the data type of *ReadBuf[]*, the data that is insufficient for the data size of *ReadBuf[]* is discarded. The file position indicator advances to the end of the file, and the value of *EOF* changes to TRUE.
- Elements beyond *Size* times *ReadBuf[]* (i.e., the elements not overwritten when data is read) will retain the values from before execution of this instruction.
- You must use the FileOpen instruction to obtain the value for *FileID* before you execute this instruction.
- Data is read in byte increments. The lower bytes are read before the upper bytes (little endian).
- A value is stored in *EOF* when the instruction is completed. Specifically, it is stored when the value of *Done* changes from FALSE to TRUE.
- If *ReadBuf[]* is an array of structures, adjustment areas between members may be inserted depending on the composition.
- If the operating mode of the CPU Unit is changed to PROGRAM mode or when a major fault level Controller error occurs during instruction execution, the file is closed by the system. Any read/write operations that are in progress are completed to the end.
- Do not simultaneously access the same file. Perform exclusive control of SD Memory Card instructions in the user program.
- You cannot specify a device variable for *ReadBuf[]*. If you specify a device variable, the data that was read is not assigned to *ReadBuf[]*.
- An error occurs in the following cases. *Error* will change to TRUE.

- The number of array elements in *ReadBuf[]* is smaller than the value of *Size*.
- The SD Memory Card is not in a usable condition.
- The file specified by *FileID* does not exist.
- The file specified by *FileID* is being accessed.
- The file specified by *FileID* was not opened in a reading mode.
- An error that prevents access occurs during SD Memory Card access.

Sample Programming

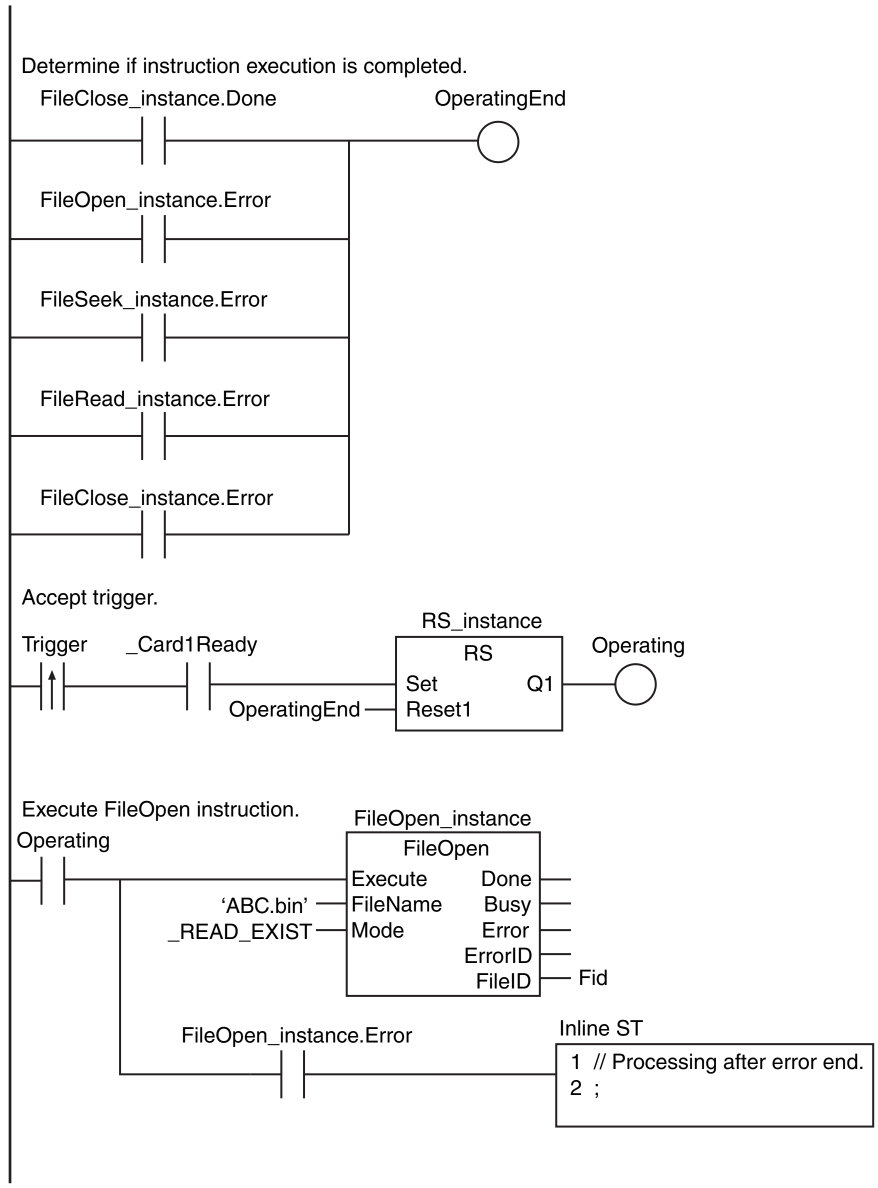
In this sample, four bytes of data are read from the second byte from beginning of the file named 'ABC.bin.' The data is written to BYTE array variable *InDat[]*. The processing procedure is as follows:

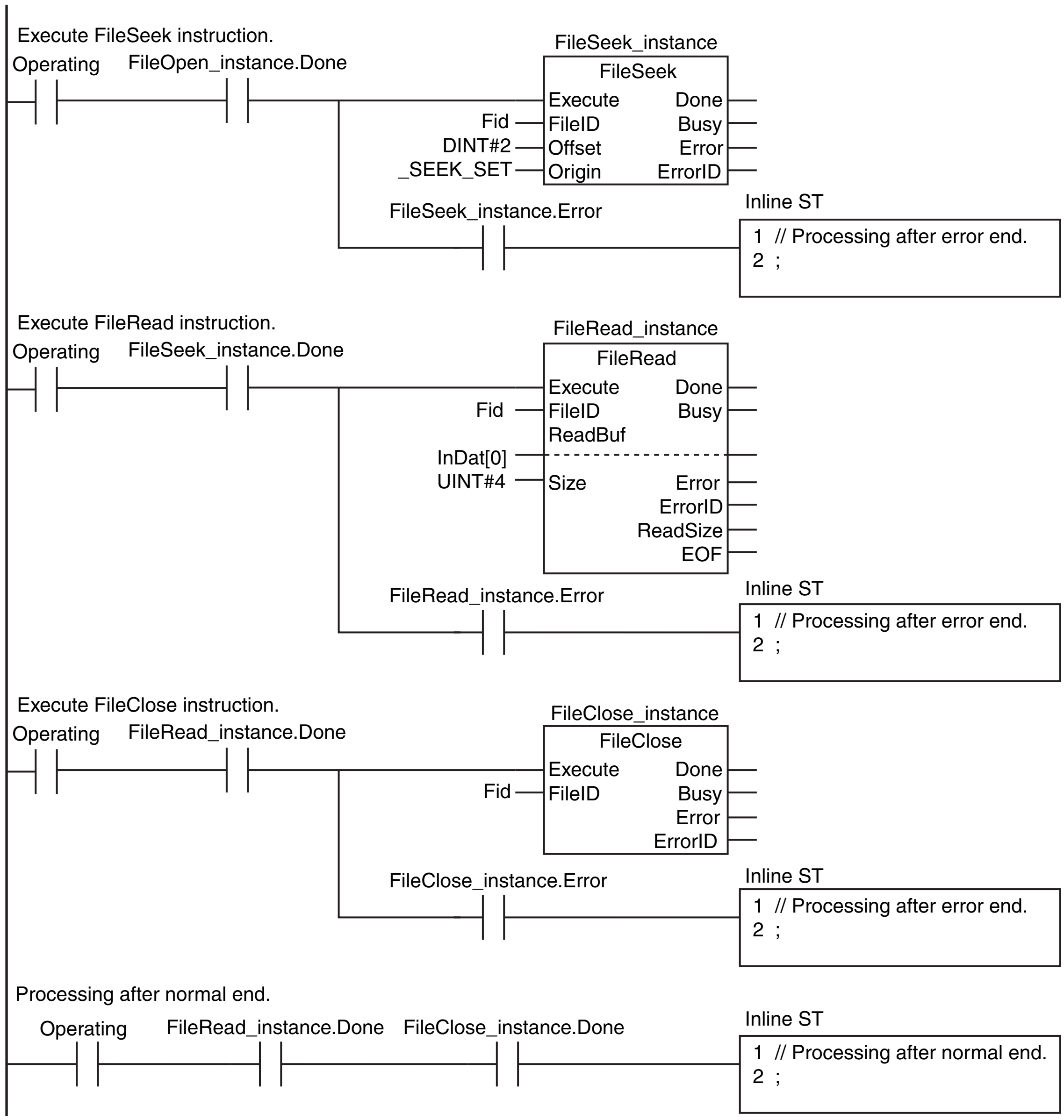
1. The FileOpen instruction is used to open the file ‘ABC.bin.’
2. The FileSeek instruction is used to set a file position indicator at the second byte from the beginning of the file.
3. The FileRead instruction is used to read four bytes of data from the position of the file position indicator and store it in array variable *InDat[]*.
4. The FileClose instruction is used to close the file ‘ABC.bin.’

LD

Internal Variables	Variable	Data type	Initial value	Comment
	OperatingEnd	BOOL	FALSE	Processing completed.
	Trigger	BOOL	FALSE	Execution condition
	Operating	BOOL	FALSE	Processing
	Fid	DWORD	16#0	File ID
	InDat	ARRAY[0..999] OF BYTE	[1000(16#0)]	Read data
	RS_instance	RS		
	FileOpen_instance	FileOpen		
	FileSeek_instance	FileSeek		
	FileRead_instance	FileRead		
	FileClose_instance	FileClose		

External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag


Determine if instruction execution is completed.
FileClose_instance.Done
OperatingEnd
FileOpen_instance.Error
FileSeek_instance.Error
FileRead_instance.Error
FileClose_instance.Error
Accept trigger.
RS_instance
Trigger
_Card1Ready
RS
Operating
Set
Q1
OperatingEnd
Reset1
Execute FileOpen instruction.
FileOpen_instance
Operating
FileOpen
Execute
Done
'ABC.bin'
FileName
Busy
_READ_EXIST
Mode
Error
ErrorID
FileID
Fid
FileOpen_instance.Error
Inline ST
1 // Processing after error end.
2 ;



Execute FileSeek instruction.
Operating
FileOpen_instance.Done
FileSeek_instance
FileSeek
Execute
FileID
Offset
Origin
Done
Busy
Error
ErrorID
Fid
DINT#2
_SEEK_SET
FileSeek_instance.Error
Inline ST
1 // Processing after error end.
2 ;
Execute FileRead instruction.
Operating
FileSeek_instance.Done
FileRead_instance
FileRead
Execute
FileID
ReadBuf
Size
Done
Busy
Error
ErrorID
ReadSize
EOF
Fid
InDat[0]
UINT#4
FileRead_instance.Error
Inline ST
1 // Processing after error end.
2 ;
Execute FileClose instruction.
Operating
FileRead_instance.Done
FileClose_instance
FileClose
Execute
FileID
Done
Busy
Error
ErrorID
Fid
FileClose_instance.Error
Inline ST
1 // Processing after error end.
2 ;
Processing after normal end.
Operating
FileRead_instance.Done
FileClose_instance.Done
Inline ST
1 // Processing after normal end.
2 ;

ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	FALSE	Execution condition
	LastTrigger	BOOL	FALSE	Value of *Trigger* from previous task period
	OperatingStart	BOOL	FALSE	Processing started.
	Operating	BOOL	FALSE	Processing
	InDat	ARRAY[0..999] OF BYTE	[1000(16#0)]	Read data
	Stage	INT	0	Stage change
	Fid	DWORD	16#0	File ID
	FileOpen_instance	FileOpen		
	FileSeek_instance	FileSeek		
	FileRead_instance	FileRead		
	FileClose_instance	FileClose		

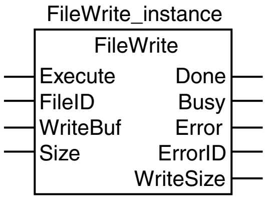
External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag

```
// Start sequence when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Card1Ready=TRUE) ) THEN
     OperatingStart:=TRUE;
     Operating     :=TRUE;
END_IF;
LastTrigger:=Trigger;

// Initialize instance.
IF (OperatingStart=TRUE) THEN
     FileOpen_instance(Execute:=FALSE);   // Initialize instance.
     FileSeek_instance(Execute:=FALSE);   // Initialize instance.
     FileRead_instance(
          Execute:=FALSE,                 // Initialize instance.
          ReadBuf:=InDat[0]);             // Dummy
     FileClose_instance(Execute:=FALSE);  // Initialize instance.
     Stage         :=INT#1;
     OperatingStart:=FALSE;
END_IF;

// Execute instructions.
IF (Operating=TRUE) THEN
     CASE Stage OF
     1 :     // Open file.
          FileOpen_instance(
               Execute:=TRUE,
               FileName:='ABC.bin',   // File name
               Mode    :=_READ_EXIST, // Read file.
               FileID   =>Fid);       // File ID

          IF (FileOpen_instance.Done=TRUE) THEN
               Stage:=INT#2;          // Normal end
          END_IF;

          IF (FileOpen_instance.Error=TRUE) THEN
               Stage:=INT#99;         // Error end
          END_IF;
```

```
        2 :     // Seek file.
             FileSeek_instance(
                  Execute:=TRUE,
                  FileID :=Fid,       // File ID
                  Offset  :=DINT#2,   // File position indicator goes to second byte from the beginning.
                  Origin :=_SEEK_SET);

             IF (FileSeek_instance.Done=TRUE) THEN
                  Stage:=INT#3;   // Normal end
             END_IF;

             IF (FileSeek_instance.Error=TRUE) THEN
                  Stage:=INT#99;  // Error end
             END_IF;

        3 :     // Read file.
             FileRead_instance(
                  Execute :=TRUE,
                  FileID  :=Fid,     // File ID
                  ReadBuf :=InDat[0],// Read buffer
                  Size    :=UINT#4); // Number of elements to read: 4 bytes

             IF (FileRead_instance.Done=TRUE) THEN
                  Stage:=INT#4;      // Normal end
             END_IF;

             IF (FileRead_instance.Error=TRUE) THEN
                  Stage:=INT#99;     // Error end
             END_IF;

        4 :     // Close file.
             FileClose_instance(
                  Execute:=TRUE,
                  FileID :=Fid);    // File ID

             IF (FileClose_instance.Done=TRUE) THEN
                  Operating:=FALSE; // Normal end
             END_IF;

             IF (FileClose_instance.Error=TRUE) THEN
                  Stage:=INT#99;    // Error end
             END_IF;

        99 :
             Operating:=FALSE;      // Processing after error end.
        END_CASE;
   END_IF;
```

FileWrite

The FileWrite instruction writes data to the specified file in the SD Memory Card.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FileWrite	Write File	FB		FileWrite_instance(Execute, FileID, WriteBuf, Size, Done, Busy, Error, ErrorID, WriteSize);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
FileID	File ID	Input	ID of file to write	Depends on data type.	---	0
WriteBuf[] (array)	Write buffer		Write data			*
Size	Number of elements to write		Number of elements to write			1
WriteSize	Number of written elements	Output	Number of elements that were actually written	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

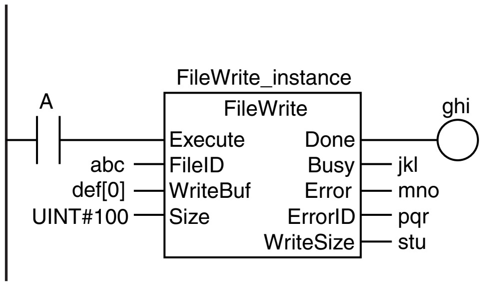
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileID				OK																
WriteBuf[] (array)	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
	Arrays of enumerations or structures can also be specified.																			
Size							OK													
WriteSize							OK													

Function

The FileWrite instruction writes data to the position of the file position indicator in the file specified by file ID *FileID* in the SD Memory Card. The file position indicator is set at the desired location in advance with the FileSeek instruction. The contents of the write buffer *WriteBuf[]* is written to the file. The amount of data that is written is the size of the data type of *WriteBuf[]* times *Size*. You can specify an array of enumerations or structures for *WriteBuf[]*. The data size that is actually written is output to *WriteSize*.

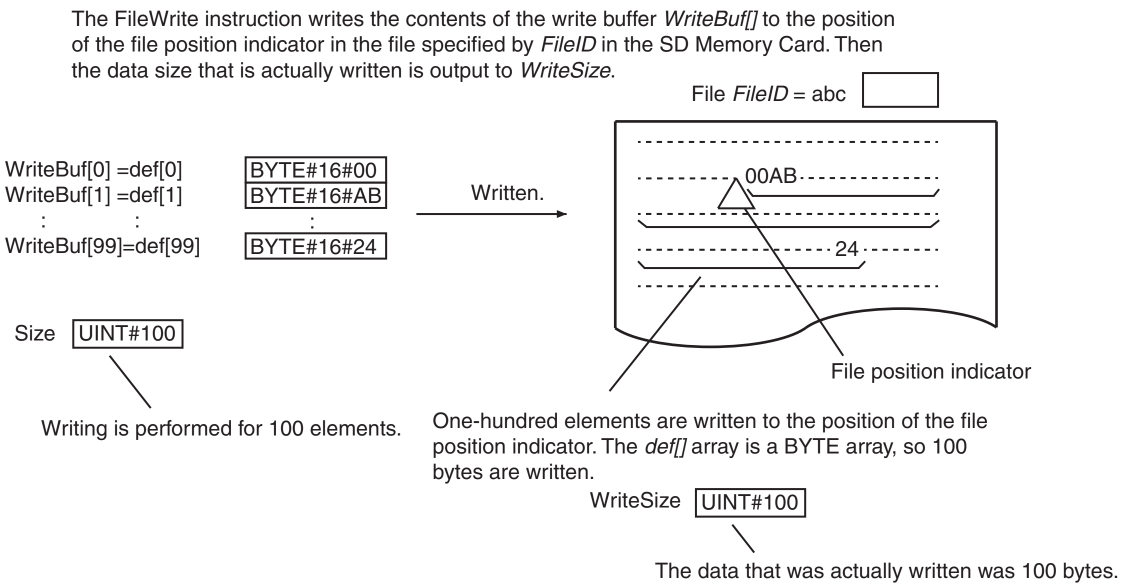
The following figure shows a programming example. If the write buffer *def[]* is BYTE data, 100 bytes of data is written to the file.

LD



ST

```
FileWrite_instance(A, abc, def[0], UINT#100, ghi,
                   jkl, mno, pqr, stu);
```



Related System-defined Variables

Name	Meaning	Data type	Description
_Card1Ready	SD Memory Card Ready Flag	BOOL	This flag indicates if the SD Memory Card can be accessed by instructions and communications commands.*1 TRUE: Can be used. FALSE: Cannot be used.
_Card1Protect*2	SD Memory Card Write Protected Flag	BOOL	This flag indicates if the SD Memory Card is write protected when it is inserted and ready to use. TRUE: Write protected. FALSE: Not write protected.

Name	Meaning	Data type	Description
_Card1Err*2	SD Memory Card Error Flag	BOOL	This flag indicates if an unspecified SD Memory Card (e.g., an SDHC card) is mounted or if the format is incorrect (i.e., not FAT16 or corrupted). TRUE: Error. FALSE: No error.
_Card1Access*2	SD Memory Card Access Flag	BOOL	This flag indicates if the SD Memory Card is currently being accessed. TRUE: Being accessed. FALSE: Not being accessed.
_Card1PowerFail	SD Memory Card Power Interruption Flag	BOOL	This flag indicates if an error occurred in completing processing when power was interrupted during access*3. This flag is not cleared automatically. TRUE: Error. FALSE: No error.

*1 For the NJ/NX-series, it is a precondition that the SD Memory Card is physically inserted and mounted normally. For an NY-series Controller, it is a precondition that the shared folder is detected by the Controller.

*2 These variables are not used for the NY-series Controller. They are fixed to FALSE.

*3 For the NJ/NX-series, this indicates an access to the SD Memory Card. For an NY-series Controller, this indicates an access to the shared folder.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- You must use the FileOpen instruction to obtain the value for *FileID* before you execute this instruction.
- Data is written in byte increments. The lower bytes are written before the upper bytes (little endian).
- If *WriteBuf[]* is an array of structures, adjustment areas between members may be inserted depending on the composition.
- If the operating mode of the CPU Unit is changed to PROGRAM mode or when a major fault level Controller error occurs during instruction execution, the file is closed by the system. Any read/write operations that are in progress are completed to the end.
- Do not simultaneously access the same file. Perform exclusive control of SD Memory Card instructions in the user program.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The number of array elements in *WriteBuf[]* is smaller than the value of Size.
  - The SD Memory Card is not in a usable condition.
  - The SD Memory Card is write protected.
  - There is insufficient space available on the SD Memory Card.
  - The file specified by *FileID* does not exist.
  - The file specified by *FileID* is being accessed.
  - The file specified by *FileID* was not opened in a writing mode.
  - An error that prevents access occurs during SD Memory Card access.

Sample Programming

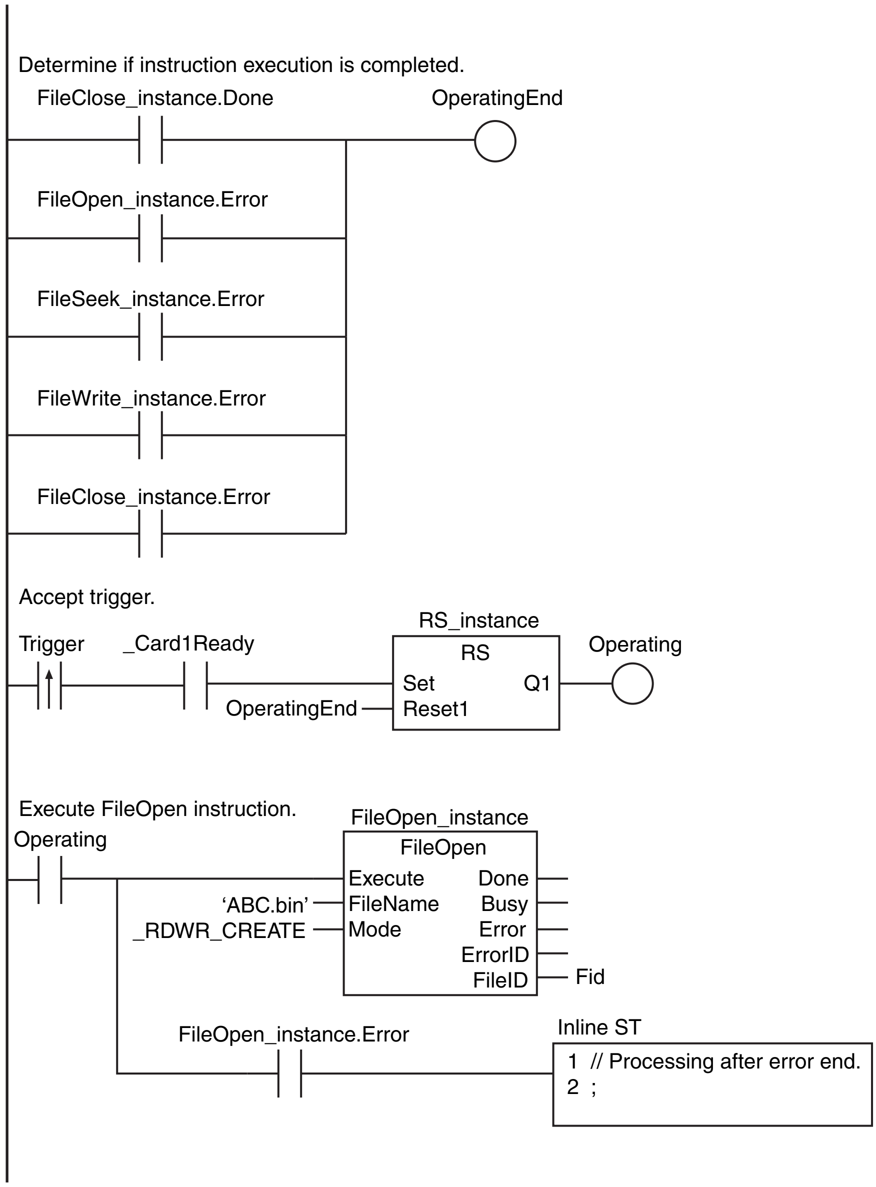
Here, four bytes of data are written from the second byte from the beginning of the file 'ABC.bin.' The contents of the BYTE array variable *OutDat[]* is written to the file. The processing procedure is as follows:

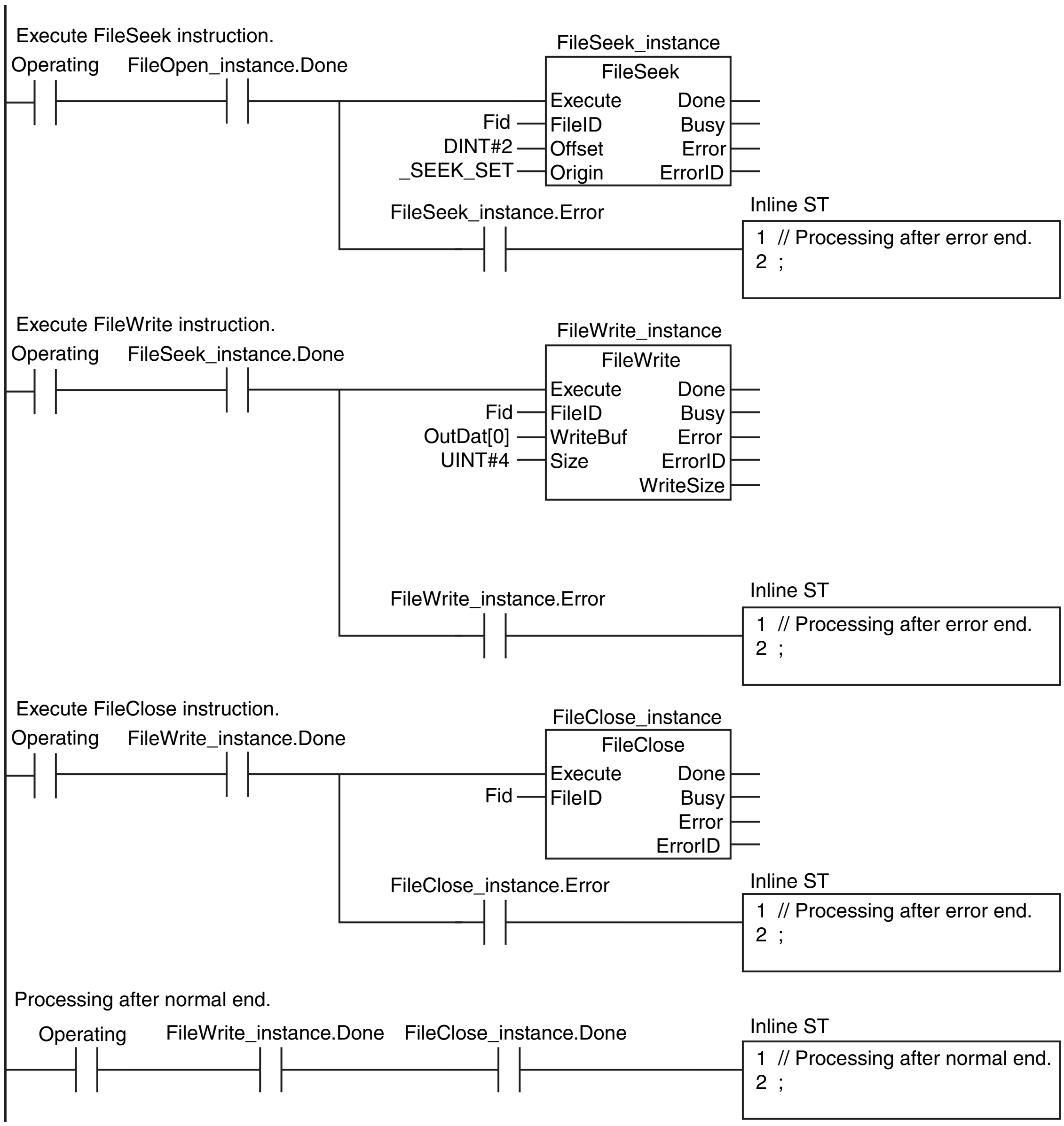
1. The FileOpen instruction is used to open the file ‘ABC.bin.’
2. The FileSeek instruction is used to set a file position indicator at the second byte from the beginning of the file.
3. The FileWrite instruction is used to write four bytes from array variable *OutDat[]* to the position of the file position indicator.
4. The FileClose instruction is used to close the file ‘ABC.bin.’

LD

Internal Variables	Variable	Data type	Initial value	Comment
	OperatingEnd	BOOL	FALSE	Processing completed.
	Trigger	BOOL	FALSE	Execution condition
	Operating	BOOL	FALSE	Processing
	Fid	DWORD	16#0	File ID
	OutDat	ARRAY[0..999] OF BYTE	[1000(16#0)]	Write data
	RS_instance	RS		
	FileOpen_instance	FileOpen		
	FileSeek_instance	FileSeek		
	FileWrite_instance	FileWrite		
	FileClose_instance	FileClose		

External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag




Execute FileSeek instruction.
Operating
FileOpen_instance.Done
FileSeek_instance
FileSeek
Execute
Done
Fid
FileID
Busy
DINT#2
Offset
Error
_SEEK_SET
Origin
ErrorID
FileSeek_instance.Error
Inline ST
1 // Processing after error end.
2 ;
Execute FileWrite instruction.
Operating
FileSeek_instance.Done
FileWrite_instance
FileWrite
Execute
Done
Fid
FileID
Busy
OutDat[0]
WriteBuf
Error
UINT#4
Size
ErrorID
WriteSize
FileWrite_instance.Error
Inline ST
1 // Processing after error end.
2 ;
Execute FileClose instruction.
Operating
FileWrite_instance.Done
FileClose_instance
FileClose
Execute
Done
Fid
FileID
Busy
Error
ErrorID
FileClose_instance.Error
Inline ST
1 // Processing after error end.
2 ;
Processing after normal end.
Operating
FileWrite_instance.Done
FileClose_instance.Done
Inline ST
1 // Processing after normal end.
2 ;

ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	FALSE	Execution condition
	LastTrigger	BOOL	FALSE	Value of *Trigger* from previous task period
	OperatingStart	BOOL	FALSE	Processing started.
	Operating	BOOL	FALSE	Processing
	OutDat	ARRAY[0..999] OF BYTE	[1000(16#0)]	Write data
	Stage	INT	0	Stage change
	Fid	DWORD	16#0	File ID
	FileOpen_instance	FileOpen		
	FileSeek_instance	FileSeek		
	FileWrite_instance	FileWrite		
	FileClose_instance	FileClose		

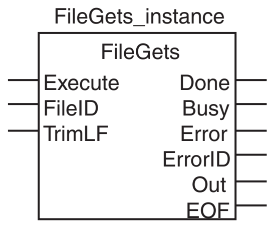
External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag

```
// Start sequence when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Card1Ready=TRUE) ) THEN
     OperatingStart:=TRUE;
     Operating     :=TRUE;
END_IF;
LastTrigger:=Trigger;

// Initialize instance.
IF (OperatingStart=TRUE) THEN
     FileOpen_instance(Execute:=FALSE);
     FileSeek_instance(Execute:=FALSE);
     FileWrite_instance(
               Execute   :=FALSE,
               WriteBuf  :=OutDat[0]);
     FileClose_instance(Execute:=FALSE);
     Stage              :=INT#1;
     OperatingStart:=FALSE;
END_IF;

// Execute instructions.
IF (Operating=TRUE) THEN
     CASE Stage OF
     1 :     // Open file.
          FileOpen_instance(
               Execute :=TRUE,
               FileName:='ABC.bin',     // File name
               Mode    :=_RDWR_CREATE, // Read file and write.
               FileID   =>Fid);        // File ID

          IF (FileOpen_instance.Done=TRUE) THEN
               Stage:=INT#2;     // Normal end
          END_IF;

          IF (FileOpen_instance.Error=TRUE) THEN
               Stage:=INT#99;    // Error end
          END_IF;
```

```
    2 :     // Seek file.
        FileSeek_instance(
            Execute :=TRUE,
            FileID  :=Fid,     // File ID
            Offset  :=DINT#2,  // File position indicator goes to second byte from the beginning.
            Origin  :=_SEEK_SET);

        IF (FileSeek_instance.Done=TRUE) THEN
            Stage:=INT#3;    // Normal end
        END_IF;

        IF (FileSeek_instance.Error=TRUE) THEN
            Stage:=INT#99;   // Error end
        END_IF;

    3 :     // Write file.
        FileWrite_instance(
            Execute :=TRUE,
            FileID  :=Fid,       // File ID
            WriteBuf:=OutDat[0], // Write buffer
            Size    :=UINT#4);   // Number of elements to write: 4 bytes

        IF (FileWrite_instance.Done=TRUE) THEN
            Stage:=INT#4;    // Normal end
        END_IF;

        IF (FileWrite_instance.Error=TRUE) THEN
            Stage:=INT#99;   // Error end
        END_IF;

    4 :     // Close file.
        FileClose_instance(
            Execute:=TRUE,
            FileID :=Fid);     // File ID

        IF (FileClose_instance.Done=TRUE) THEN
            Operating:=FALSE;  // Normal end
        END_IF;

        IF (FileClose_instance.Error=TRUE) THEN
            Stage:=INT#99;    // Error end
        END_IF;

    99 :
        Operating:=FALSE;      // Processing after error end.
    END_CASE;
END_IF;
```

FileGets

The FileGets instruction reads a text string of one line from the specified file in the SD Memory Card.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FileGets	Get Text String	FB		FileGets_instance(Execute, FileID, TrimLF, Done, Busy, Error, ErrorID, Out, EOF);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
FileID	File ID	Input	ID of file to read	Depends on data type.	---	0
TrimLF	Line feed designation		Handling of the line feed code of text string that was read TRUE: Delete. FALSE: Do not delete.			FALSE
Out	Read text string	Output	Text string that was read	Depends on data type.	---	---
EOF	End of file		Whether end of file was reached TRUE: Reached. FALSE: Not reached.			

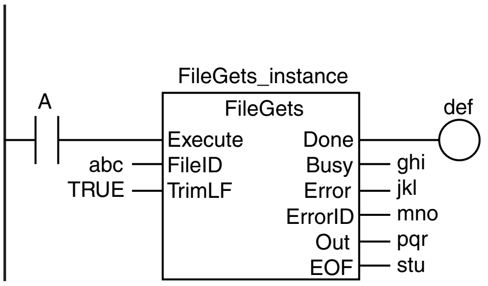
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileID				OK																
TrimLF	OK																			
Out																				OK
EOF	OK																			

Function

The FileGets instruction reads a text string of one line from the position of the file position indicator in the file specified by file ID *FileID* in the SD Memory Card. The file position indicator is set at the desired location in advance with the FileSeek instruction. Line endings are determined by a line feed code. The text string that is read is written to read text string *Out*. The following three line feeds are automatically detected: CR, LF, and CR+LF. If line feed designation *TrimLF* is TRUE, the line feed code is deleted from the text string before it is written to *Out*. If data is read to the end of the file, end of file *EOF* changes to TRUE. Otherwise, the value of *EOF* will be FALSE.

The following figure shows a programming example. Here, a text string of one line is read from a file, the line feed code is deleted, and the result is written to *pqr*.

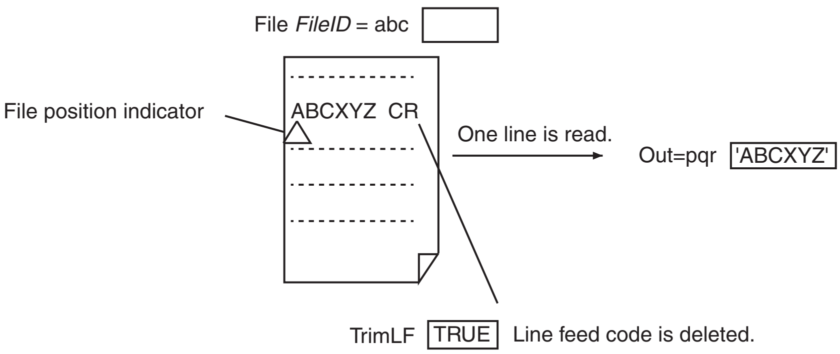
LD



ST

```
FileGets_instance(A, abc, TRUE, def, ghi,
                  jkl, mno, pqr, stu);
```

The FileGets instruction reads a text string of one line from the position of the file position indicator in the file specified by *FileID* in the SD Memory Card and stores it in the read text string *Out*. The line feed code is deleted.



Related System-defined Variables

Name	Meaning	Data type	Description
_Card1Ready	SD Memory Card Ready Flag	BOOL	This flag indicates if the SD Memory Card can be accessed by instructions and communications commands.*1 TRUE: Can be used. FALSE: Cannot be used.
_Card1Protect*2	SD Memory Card Write Protected Flag	BOOL	This flag indicates if the SD Memory Card is write protected when it is inserted and ready to use. TRUE: Write protected. FALSE: Not write protected.

Name	Meaning	Data type	Description
_Card1Err*2	SD Memory Card Error Flag	BOOL	This flag indicates if an unspecified SD Memory Card (e.g., an SDHC card) is mounted or if the format is incorrect (i.e., not FAT16 or corrupted). TRUE: Error. FALSE: No error.
_Card1Access*2	SD Memory Card Access Flag	BOOL	This flag indicates if the SD Memory Card is currently being accessed. TRUE: Being accessed. FALSE: Not being accessed.
_Card1PowerFail	SD Memory Card Power Interruption Flag	BOOL	This flag indicates if an error occurred in completing processing when power was interrupted during access*3. This flag is not cleared automatically. TRUE: Error. FALSE: No error.

*1 For the NJ/NX-series, it is a precondition that the SD Memory Card is physically inserted and mounted normally. For an NY-series Controller, it is a precondition that the shared folder is detected by the Controller.

*2 These variables are not used for the NY-series Controller. They are fixed to FALSE.

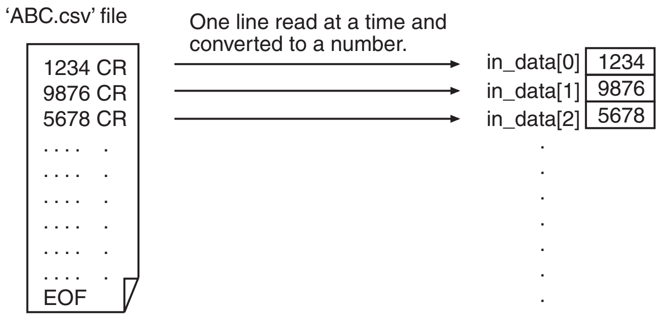
*3 For the NJ/NX-series, this indicates an access to the SD Memory Card. For an NY-series Controller, this indicates an access to the shared folder.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- If the length of the one-line text string exceeds 1,986 bytes (with UTF-8 character codes, including the final NULL character), the first 1,985 bytes of the text string are stored in *Out* with a NULL character attached.
- You must use the FileOpen instruction to obtain the value for *FileID* before you execute this instruction.
- If the operating mode of the CPU Unit is changed to PROGRAM mode or when a major fault level Controller error occurs during instruction execution, the file is closed by the system. Any read/write operations that are in progress are completed to the end.
- Do not simultaneously access the same file. Perform exclusive control of SD Memory Card instructions in the user program.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The SD Memory Card is not in a usable condition.
  - The file specified by *FileID* does not exist.
  - The file specified by *FileID* is being accessed.
  - The file specified by *FileID* was not opened in a reading mode.
  - An error that prevents access occurs during SD Memory Card access.

Sample Programming

Here, multiple text strings that are separated by CR codes are stored in a file named ‘ABC.csv.’ All of them are text strings of numbers. One line at a time is read from the file, the text strings are converted to integers, and the results are stored in INT array variable *InDat[]*. Processing is ended when all of the data to the end of the file is read.
It is assumed that this sample programming is in a periodic task.



The processing procedure is as follows:

1 The FileOpen instruction is used to open the file ‘ABC.csv.’

2 The FileGets instruction is used to read one line from the file.

3 The STRING_TO_INT instruction is used to convert the text string that was read to an integer and store it in *InDat[]*.

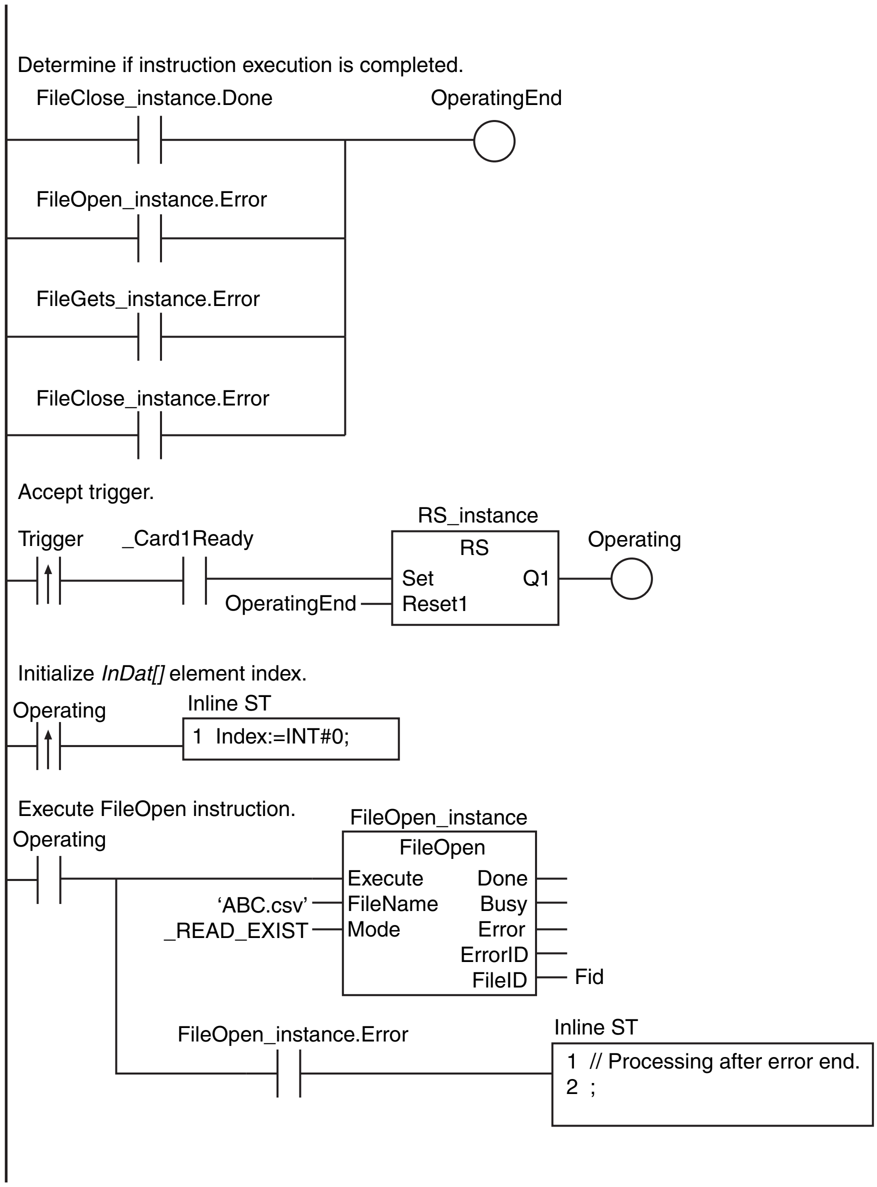
4 Steps 2 and 3 are repeated until the EOF (end of file).

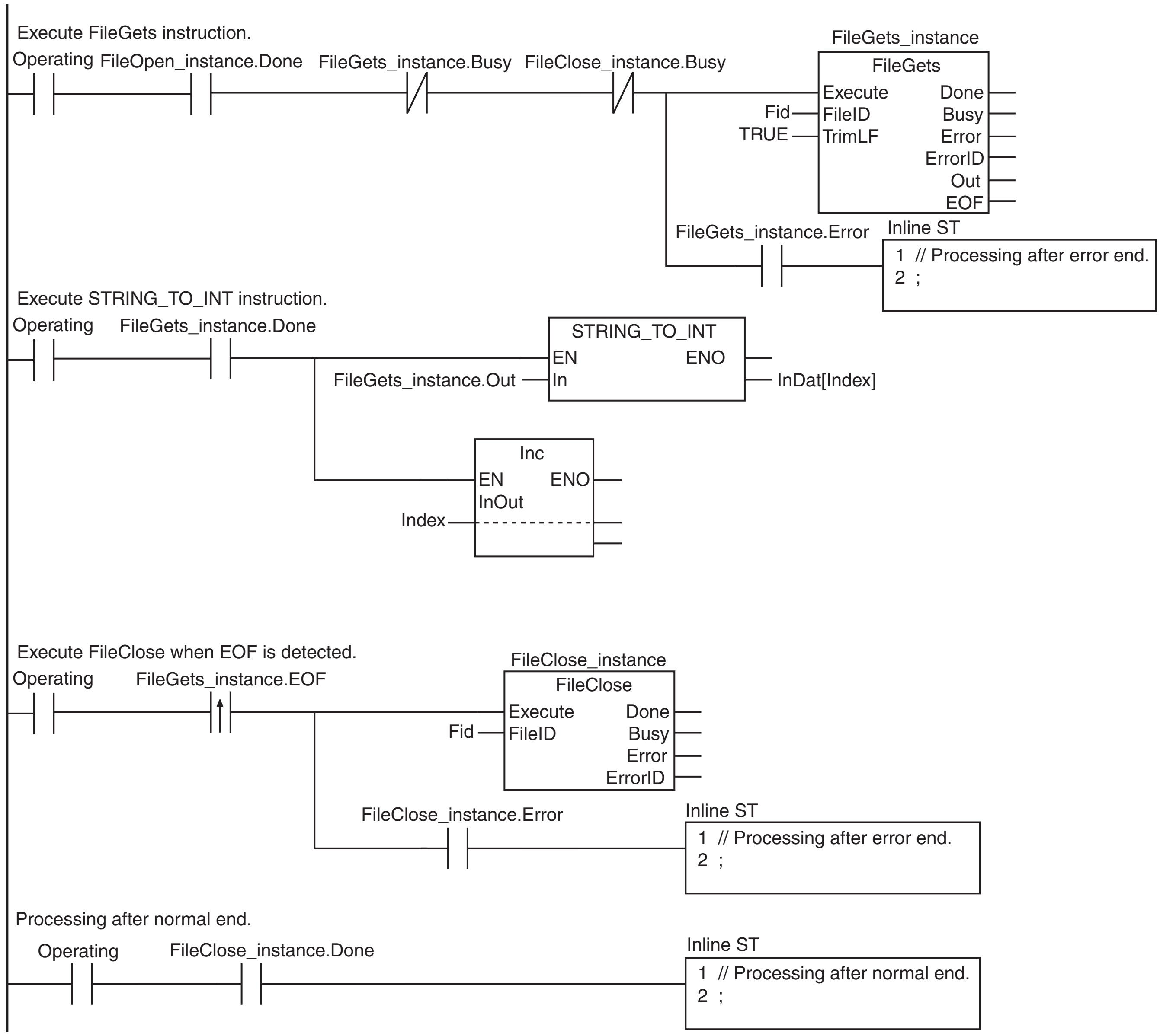
5 The FileClose instruction is used to close the file.

LD

Internal Variables	Variable	Data type	Initial value	Comment
	OperatingEnd	BOOL	FALSE	Processing completed.
	Trigger	BOOL	FALSE	Execution condition
	Operating	BOOL	FALSE	Processing
	Index	INT	0	*InDat[]* element index
	Fid	DWORD	16#0	File ID
	InDat	ARRAY[0..999] OF INT	[1000(0)]	Integer data
	RS_instance	RS		
	FileOpen_instance	FileOpen		
	FileGets_instance	FileGets		
	FileClose_instance	FileClose		

External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag




Execute FileGets instruction.
Operating
FileOpen_instance.Done
FileGets_instance.Busy
FileClose_instance.Busy
FileGets_instance
FileGets
Execute
Done
Fid
FileID
Busy
TRUE
TrimLF
Error
ErrorID
Out
EOF
FileGets_instance.Error
Inline ST
1 // Processing after error end.
2 ;
Execute STRING_TO_INT instruction.
Operating
FileGets_instance.Done
STRING_TO_INT
EN
ENO
FileGets_instance.Out
In
InDat[Index]
Inc
EN
ENO
InOut
Index
Execute FileClose when EOF is detected.
Operating
FileGets_instance.EOF
FileClose_instance
FileClose
Execute
Done
Fid
FileID
Busy
Error
ErrorID
FileClose_instance.Error
Inline ST
1 // Processing after error end.
2 ;
Processing after normal end.
Operating
FileClose_instance.Done
Inline ST
1 // Processing after normal end.
2 ;

ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	FALSE	Execution condition
	LastTrigger	BOOL	FALSE	Value of *Trigger* from previous task period
	OperatingStart	BOOL	FALSE	Processing started.
	Operating	BOOL	FALSE	Processing
	InDat	ARRAY[0..999] OF INT	[1000(0)]	Integer data
	Stage	INT	0	Stage change
	Index	INT	0	*InDat[]* element index
	Fid	DWORD	16#0	File ID
	FileOpen_instance	FileOpen		
	FileGets_instance	FileGets		
	FileClose_instance	FileClose		

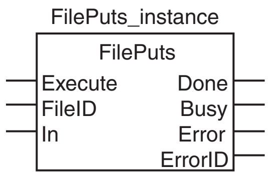
External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag

```
// Start sequence when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Card1Ready=TRUE) ) THEN
    OperatingStart:=TRUE;
    Operating     :=TRUE;
END_IF;
LastTrigger:=Trigger;

// Initialize instance.
IF (OperatingStart=TRUE) THEN
    FileOpen_instance(Execute:=FALSE);
    FileGets_instance(Execute:=FALSE);
    FileClose_instance(Execute:=FALSE);
    Stage :=INT#1;
    Index :=INT#0;
    OperatingStart:=FALSE;
END_IF;

// Execute instructions.
IF (Operating=TRUE) THEN
    CASE Stage OF
    1 :     // Open file.
        FileOpen_instance(
            Execute:=TRUE,
            FileName:='ABC.csv',    // File name
            Mode    :=_READ_EXIST,  // Read file.
            FileID   =>Fid);        // File ID

        IF (FileOpen_instance.Done=TRUE) THEN
            Stage:=INT#2;   // Normal end
        END_IF;

        IF (FileOpen_instance.Error=TRUE) THEN
            Stage:=INT#99;  // Error end
        END_IF;

    2 :     // Read text string.
        FileGets_instance(
            Execute:=TRUE,
```

```
            FileID :=Fid,
            TrimLF :=TRUE);

        IF (FileGets_instance.Done=TRUE) THEN
            // Convert the text string that was read to an integer.
            InDat[Index]:=STRING_TO_INT(FileGets_instance.Out);
            Index:=Index+INT#1;

            // Reached end of file.
            IF (FileGets_instance.EOF=TRUE) THEN
                Stage:=INT#3;   // Normal end
            ELSE
                    FileGets_instance(Execute:=FALSE);
            END_IF;
        END_IF;

        IF (FileGets_instance.Error=TRUE) THEN
            Stage:=INT#99;       // Error end
        END_IF;

     3 :     // Close file.
        FileClose_instance(
            Execute:=TRUE,
            FileID :=Fid);    // File ID

        IF (FileClose_instance.Done=TRUE) THEN
            Operating:=FALSE; // Normal end
        END_IF;

        IF (FileClose_instance.Error=TRUE) THEN
            Stage:=INT#99;    // Error end
        END_IF;

    99 :     // Processing after error end.
        Operating:=FALSE;
    END_CASE;
END_IF;
```

FilePuts

The FilePuts instruction writes a text string to the specified file in the SD Memory Card.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FilePuts	Put Text String	FB		FilePuts_instance(Execute, FileID, In, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
FileID	File ID	Input	ID of file to write	Depends on data type.	---	0
In	Write text string		Text string to write			''

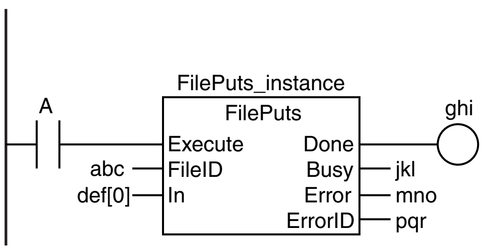
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileID				OK																
In																				OK

Function

The FilePuts instruction writes a text string to the position of the file position indicator in the file specified by file ID *FileID* in the SD Memory Card. The file position indicator is set at the desired location in advance with the FileSeek instruction. The contents of write text string *In* is written to the file.

The following figure shows a programming example. Here, the contents of array element *def[0]* is written to the file.

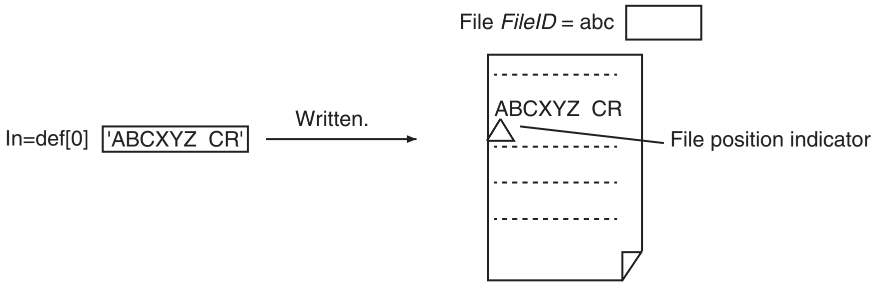
LD



ST

```
FilePuts_instance(A, abc, def[0], ghi, jkl, mno, pqr);
```

The FilePuts instruction writes the contents of the write text string *In* to the position of the file position indicator in the file specified by *FileID* in the SD Memory Card.



Related System-defined Variables

Name	Meaning	Data type	Description
_Card1Ready	SD Memory Card Ready Flag	BOOL	This flag indicates if the SD Memory Card can be accessed by instructions and communications commands.*1 TRUE: Can be used. FALSE: Cannot be used.
_Card1Protect*2	SD Memory Card Write Protected Flag	BOOL	This flag indicates if the SD Memory Card is write protected when it is inserted and ready to use. TRUE: Write protected. FALSE: Not write protected.
_Card1Err*2	SD Memory Card Error Flag	BOOL	This flag indicates if an unspecified SD Memory Card (e.g., an SDHC card) is mounted or if the format is incorrect (i.e., not FAT16 or corrupted). TRUE: Error. FALSE: No error.
_Card1Access*2	SD Memory Card Access Flag	BOOL	This flag indicates if the SD Memory Card is currently being accessed. TRUE: Being accessed. FALSE: Not being accessed.
_Card1PowerFail	SD Memory Card Power Interruption Flag	BOOL	This flag indicates if an error occurred in completing processing when power was interrupted during access*3. This flag is not cleared automatically. TRUE: Error. FALSE: No error.

*1 For the NJ/NX-series, it is a precondition that the SD Memory Card is physically inserted and mounted normally. For an NY-series Controller, it is a precondition that the shared folder is detected by the Controller.

*2 These variables are not used for the NY-series Controller. They are fixed to FALSE.

*3 For the NJ/NX-series, this indicates an access to the SD Memory Card. For an NY-series Controller, this indicates an access to the shared folder.

Additional Information

To create a line feed after you write the text sting, add a line feed code to the end of *In*.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- You must use the FileOpen instruction to obtain the value for *FileID* before you execute this instruction.
- If the operating mode of the CPU Unit is changed to PROGRAM mode or when a major fault level Controller error occurs during instruction execution, the file is closed by the system. Any read/write operations that are in progress are completed to the end.
- Do not simultaneously access the same file. Perform exclusive control of SD Memory Card instructions in the user program.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The SD Memory Card is not in a usable condition.
  - The SD Memory Card is write protected.
  - There is insufficient space available on the SD Memory Card.
  - The file specified by *FileID* does not exist.
  - The file specified by *FileID* is being accessed.
  - The file specified by *FileID* was not opened in a writing mode.
  - An error that prevents access occurs during SD Memory Card access.

Sample Programming

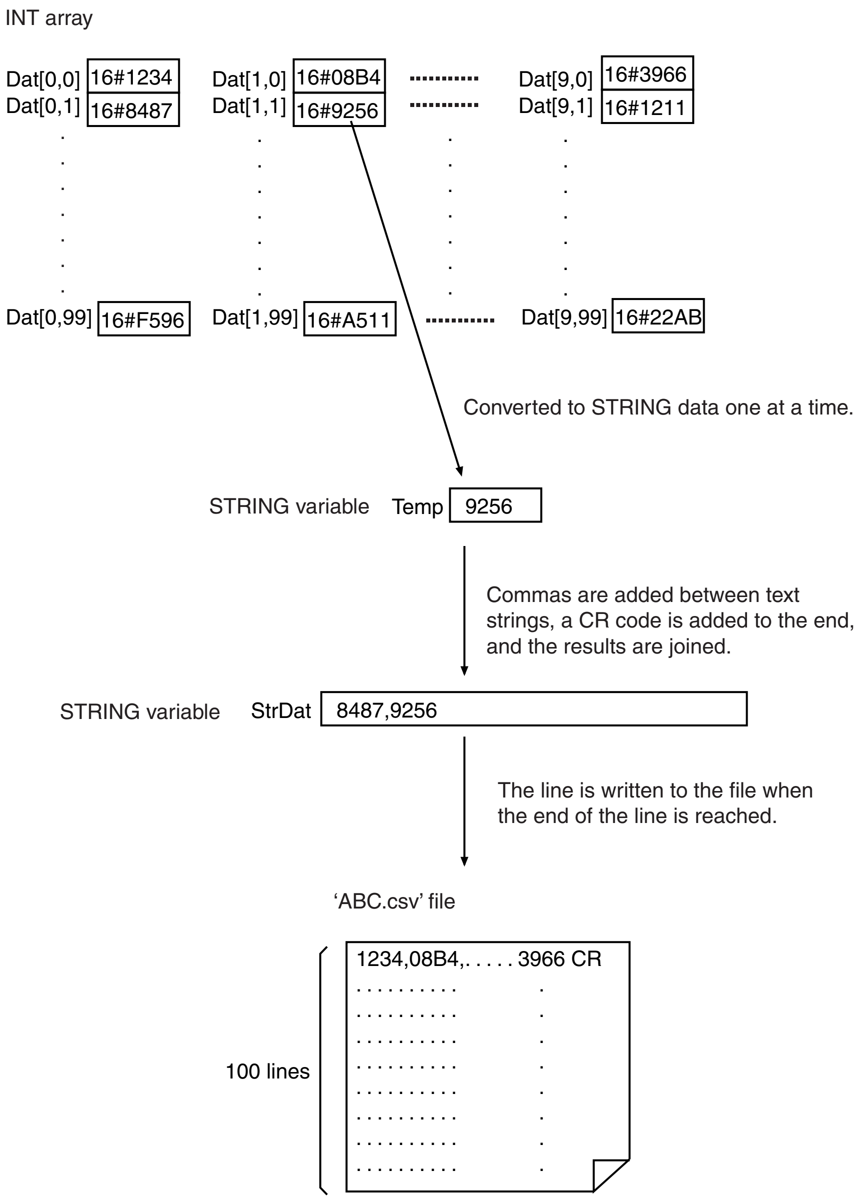
Here, 100 lines of the contents of INT array variable *Dat[0..9,0..99]* are stored in a file named 'ABC.csv' in CSV file format. Each line contains ten text strings of numbers. Commas are inserted between them. A CR+LF code is added to the end of the line. The procedure is as follows:

1 One element of *Dat[]* is converted to one text string and stored in the STRING variable *Temp*.

2 Except at the end of a line, a comma is added to the end of *Temp*. At the end of the line, a CR+LF code is added to the end of *Temp*. These are joined in the STRING variable *StrDat*.

3 When the end of the line is reached, *StrDat* is written to the file.

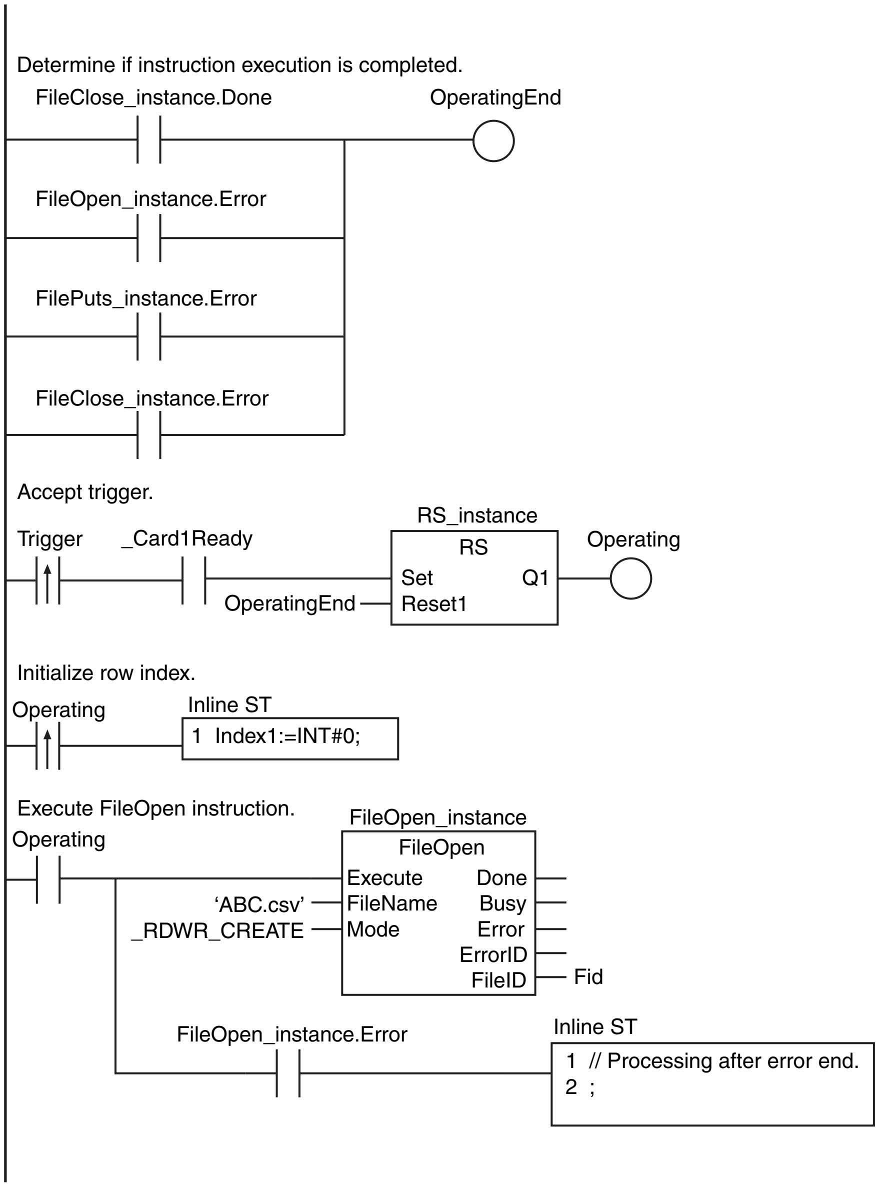
4 Steps 1 to 3 are repeated for 100 lines.

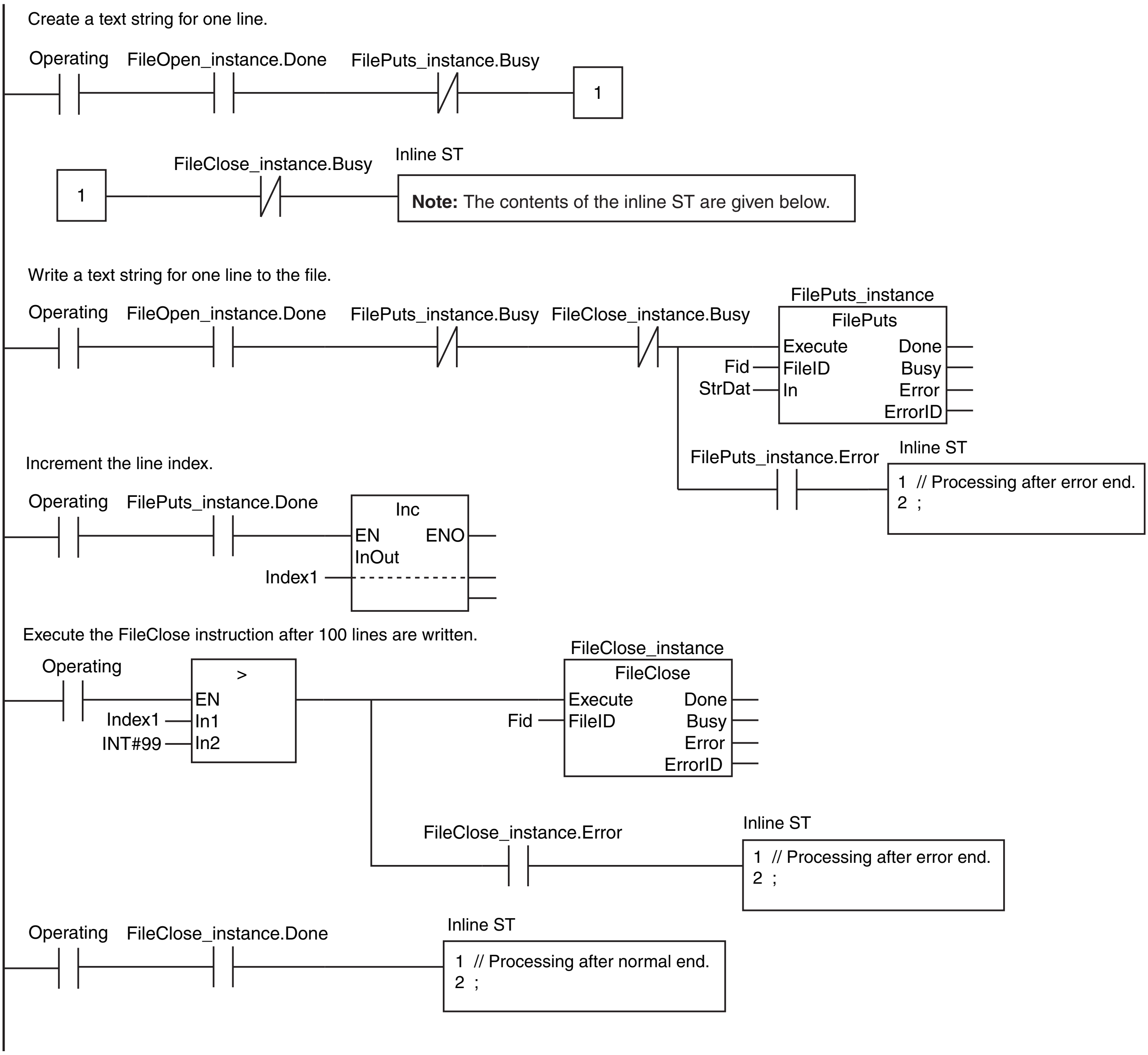


LD

Internal Variables	Variable	Data type	Initial value	Comment
	OperatingEnd	BOOL	FALSE	Processing completed.
	Trigger	BOOL	FALSE	Execution condition
	Operating	BOOL	FALSE	Processing
	Index0	INT	0	Column index
	Index1	INT	0	Row index
	Fid	DWORD	16#0	File ID
	StrDat	STRING[255]	''	Text string data
	Dat	ARRAY[0..99,0..9] OF INT	[1000(0)]	Numeric data
	Temp	STRING[255]	''	Temporary data
	RS_instance	RS		
	FileOpen_instance	FileOpen		
	FilePuts_instance	FilePuts		
	FileClose_instance	FileClose		

External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag


Determine if instruction execution is completed.
FileClose_instance.Done
OperatingEnd
FileOpen_instance.Error
FilePuts_instance.Error
FileClose_instance.Error
Accept trigger.
Trigger
_Card1Ready
RS_instance
RS
Set
Q1
OperatingEnd
Reset1
Operating
Initialize row index.
Operating
Inline ST
1 Index1:=INT#0;
Execute FileOpen instruction.
Operating
FileOpen_instance
FileOpen
Execute
Done
'ABC.csv'
FileName
Busy
_RDWR_CREATE
Mode
Error
ErrorID
FileID
Fid
FileOpen_instance.Error
Inline ST
1 // Processing after error end.
2 ;



Contents of Inline ST

```
StrDat:='';

// Concatenate text strings 0 to 8.
FOR Index0:=INT#0 TO INT#8 BY INT#1 DO
        Temp  :=INT_TO_STRING(Dat[Index1, Index0]);
        Temp  :=CONCAT(In1:=Temp, In2:=',');
        StrDat:=CONCAT(In1:=StrDat, In2:=Temp);
END_FOR;

// Concatenate text string 9 and add CR+LF.
Temp  :=INT_TO_STRING(Dat[Index1, Index0]);
Temp  :=CONCAT(In1:=Temp, In2:='$r$l');   // CR+LF
StrDat:=CONCAT(In1:=StrDat, In2:=Temp);
```

ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	FALSE	Execution condition
	LastTrigger	BOOL	FALSE	Value of *Trigger* from previous task period
	OperatingStart	BOOL	FALSE	Processing started.
	Operating	BOOL	FALSE	Processing
	Stage	INT	0	Stage change
	Index0	INT	0	Column index
	Index1	INT	0	Row index
	Fid	DWORD	16#0	File ID
	StrDat	STRING[255]	''	Text string data
	Dat	ARRAY[0..99,0..9] OF INT	[1000(0)]	Numeric data
	Temp	STRING[255]	''	Temporary data
	FileOpen_instance	FileOpen		
	FilePuts_instance	FilePuts		
	FileClose_instance	FileClose		

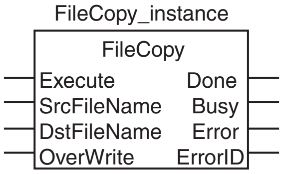
External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag

```
// Start sequence when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Card1Ready=TRUE) ) THEN
    OperatingStart:=TRUE;
    Operating     :=TRUE;
END_IF;
LastTrigger:=Trigger;

// Initialize instance.
IF (OperatingStart=TRUE) THEN
    FileOpen_instance(Execute:=FALSE);
    FilePuts_instance(Execute:=FALSE);
    FileClose_instance(Execute:=FALSE);
    Stage  :=INT#1;
    Index1 :=INT#0;  // Initialize row index.
    OperatingStart:=FALSE;
END_IF;

// Execute instructions.
IF (Operating=TRUE) THEN
    CASE Stage OF
    1 :     // Open file.
        FileOpen_instance(
            Execute :=TRUE,
            FileName:='ABC.csv',     // File name
            Mode    :=_RDWR_CREATE, // Read file
            FileID   =>Fid);         // File ID

        IF (FileOpen_instance.Done=TRUE) THEN
            Stage:=INT#2;    // Normal end
        END_IF;

        IF (FileOpen_instance.Error=TRUE) THEN
            Stage:=INT#99;   // Error end
        END_IF;
```

```
    2 :     // Create a text string for one line.
        StrDat:='';

        // Concatenate text strings 0 to 8.
        FOR Index0 :=INT#0 TO INT#8 BY INT#1 DO
            Temp  :=INT_TO_STRING(Dat[Index1, Index0]);
            Temp  :=CONCAT(In1:=Temp, In2:=',');
            StrDat:=CONCAT(In1:=StrDat, In2:=Temp);
        END_FOR;

        // Concatenate text string 9 and add CR+LF.
        Temp  :=INT_TO_STRING(Dat[Index1, Index0]);
        Temp  :=CONCAT(In1:=Temp, In2:='$r$l');
        StrDat:=CONCAT(In1:=StrDat, In2:=Temp);

        Stage:=INT#3;

    3 :     // Write text string.
        FilePuts_instance(
            Execute:=TRUE,
            FileID :=Fid,
            In     :=StrDat);

        IF (FilePuts_instance.Done=TRUE) THEN
            Index1:=Index1+INT#1;

            IF (Index1>INT#99) THEN   // If 100 lines were written...
                Stage:=INT#4;
            ELSE
                FilePuts_instance(Execute:=FALSE);
                Stage:=INT#2;
            END_IF;
        END_IF;

        IF (FilePuts_instance.Error=TRUE) THEN
            Stage:=INT#99;    // Error end
        END_IF;

    4 :     // Close file.
        FileClose_instance(
            Execute:=TRUE,
            FileID :=Fid);    // File ID

        IF (FileClose_instance.Done=TRUE) THEN
            Operating:=FALSE; // Normal end
        END_IF;

        IF (FileClose_instance.Error=TRUE) THEN
            Stage:=INT#99;    // Error end
        END_IF;

    99 :     // Processing after error end.
        Operating:=FALSE;
    END_CASE;
END_IF;
```

FileCopy

The FileCopy instruction copies the specified file in the SD Memory Card.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FileCopy	Copy File	FB		FileCopy_instance(Execute, SrcFileName, DstFileName, OverWrite, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
SrcFile Name	Source file	Input	Name of file to copy	66 bytes max. (65 single-byte alphanumeric characters plus the final NULL character)	---	''
DstFile Name	Destination file		Name of destination file			
OverWrite	Overwrite enable		TRUE: Enable overwrite. FALSE: Prohibit overwrite.	Depends on data type.		FALSE

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
SrcFile Name																				OK
DstFile Name																				OK
OverWrite	OK																			

Function

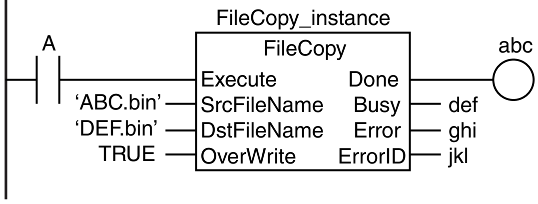
The FileCopy instruction copies the file specified by source file *SrcFileName* to designation file *DstFileName* in the SD Memory Card.

If a file with the name *DstFileName* already exists in the SD Memory Card, the following processing is performed depending on the value of overwrite enable *OverWrite*.

Value of *OverWrite*	Treatment
TRUE (Enable overwrite.)	The existing file is overwritten.
FALSE (Prohibit overwrite.)	The file is not overwritten and an error occurs.

The following figure shows a programming example. Here, the file ‘DEF.bin’ is overwritten with the file ‘ABC.bin.’

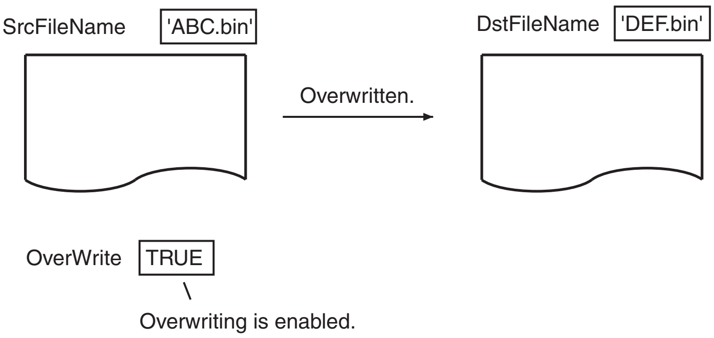
LD



ST

```
FileCopy_instance(A, ’ABC.bin’, ‘DEF.bin’,
                  TRUE, abc, def, ghi, jkl);
```

The FileCopy instruction overwrites the file specified by source file *SrcFileName* to designation file *DstFileName* in the SD Memory Card.



Related System-defined Variables

Name	Meaning	Data type	Description
_Card1Ready	SD Memory Card Ready Flag	BOOL	This flag indicates if the SD Memory Card can be accessed by instructions and communications commands.*1 TRUE: Can be used. FALSE: Cannot be used.
_Card1Protect*2	SD Memory Card Write Protected Flag	BOOL	This flag indicates if the SD Memory Card is write protected when it is inserted and ready to use. TRUE: Write protected. FALSE: Not write protected.
_Card1Err*2	SD Memory Card Error Flag	BOOL	This flag indicates if an unspecified SD Memory Card (e.g., an SDHC card) is mounted or if the format is incorrect (i.e., not FAT16 or corrupted). TRUE: Error. FALSE: No error.
_Card1Access*2	SD Memory Card Access Flag	BOOL	This flag indicates if the SD Memory Card is currently being accessed. TRUE: Being accessed. FALSE: Not being accessed.
_Card1PowerFail	SD Memory Card Power Interruption Flag	BOOL	This flag indicates if an error occurred in completing processing when power was interrupted during access*3. This flag is not cleared automatically. TRUE: Error. FALSE: No error.

*1 For the NJ/NX-series, it is a precondition that the SD Memory Card is physically inserted and mounted normally. For an NY-series Controller, it is a precondition that the shared folder is detected by the Controller.

*2 These variables are not used for the NY-series Controller. They are fixed to FALSE.

*3 For the NJ/NX-series, this indicates an access to the SD Memory Card. For an NY-series Controller, this indicates an access to the shared folder.

Additional Information

The root directory of the file name is the top level of the SD Memory Card.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- If the copy operation fails, the file specified by *DstFileName* may remain in an incomplete state in the SD Memory Card.
- If a file is open when the operating mode of the CPU Unit is changed to PROGRAM mode or when a major fault level Controller error occurs, the file is closed by the system. Any read/write operations that are in progress are completed to the end.
- For an NJ/NX-series CPU Unit, if a file is open when the power supply is stopped with the SD Memory Card power supply switch, the file is not corrupted.
- For an NJ/NX-series CPU Unit, if a file is open and the SD Memory Card is removed before the SD Memory Card power supply switch is pressed, the contents of the file will sometimes be corrupted. Always turn OFF the power supply before removing the SD Memory Card.
- For an NJ/NX-series CPU Unit, if a file is open when the power supply is stopped or the SD Memory Card is removed, it will not be possible to read or write the file even if the SD Memory Card is inserted again.
- Do not simultaneously access the same file. Perform exclusive control of SD Memory Card instructions in the user program.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The SD Memory Card is not in a usable condition.
  - The SD Memory Card is write protected.
  - There is insufficient space available on the SD Memory Card.
  - The file specified by *SrcFileName* does not exist.
  - The value of *SrcFileName* is not a valid file name.
  - The value of *DstFileName* is not a valid file name.
  - The maximum number of files or directories is exceeded.
  - The file specified by *SrcFileName* or *DstFileName* is already being accessed.
  - A file with the name *DstFileName* already exits and the value of *OverWrite* is FALSE.
  - A file with the name *DstFileName* already exits and the file is write protected.
  - If more than four SD Memory Card instructions that do not have a *FileID* variable (i.e., FileWriteVar, FileReadVar, FileCopy, DirCreate, FileRemove, DirRemove, and FileRename) are executed at the same time.
  - The value of *DstFileName* exceeds the maximum number of bytes allowed in a file name.
  - An error that prevents access occurs during SD Memory Card access.

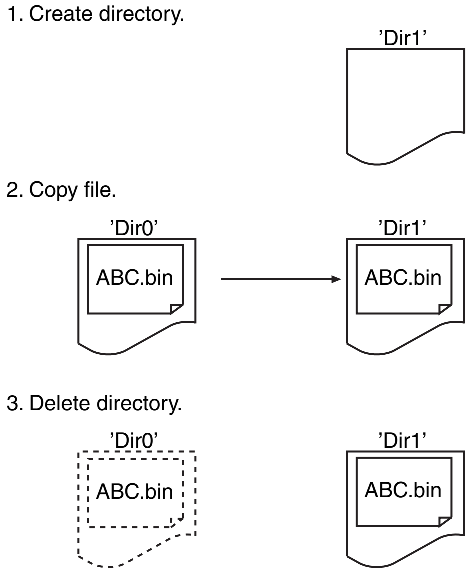
Sample Programming

The following procedure is used to move a file.

1 The DirCreate instruction is used to create a directory called 'Dir1' in the SD Memory Card.

2 The FileCopy instruction is used to copy the file named 'ABC.bin' in the existing directory 'Dir0' to the directory 'Dir1.'

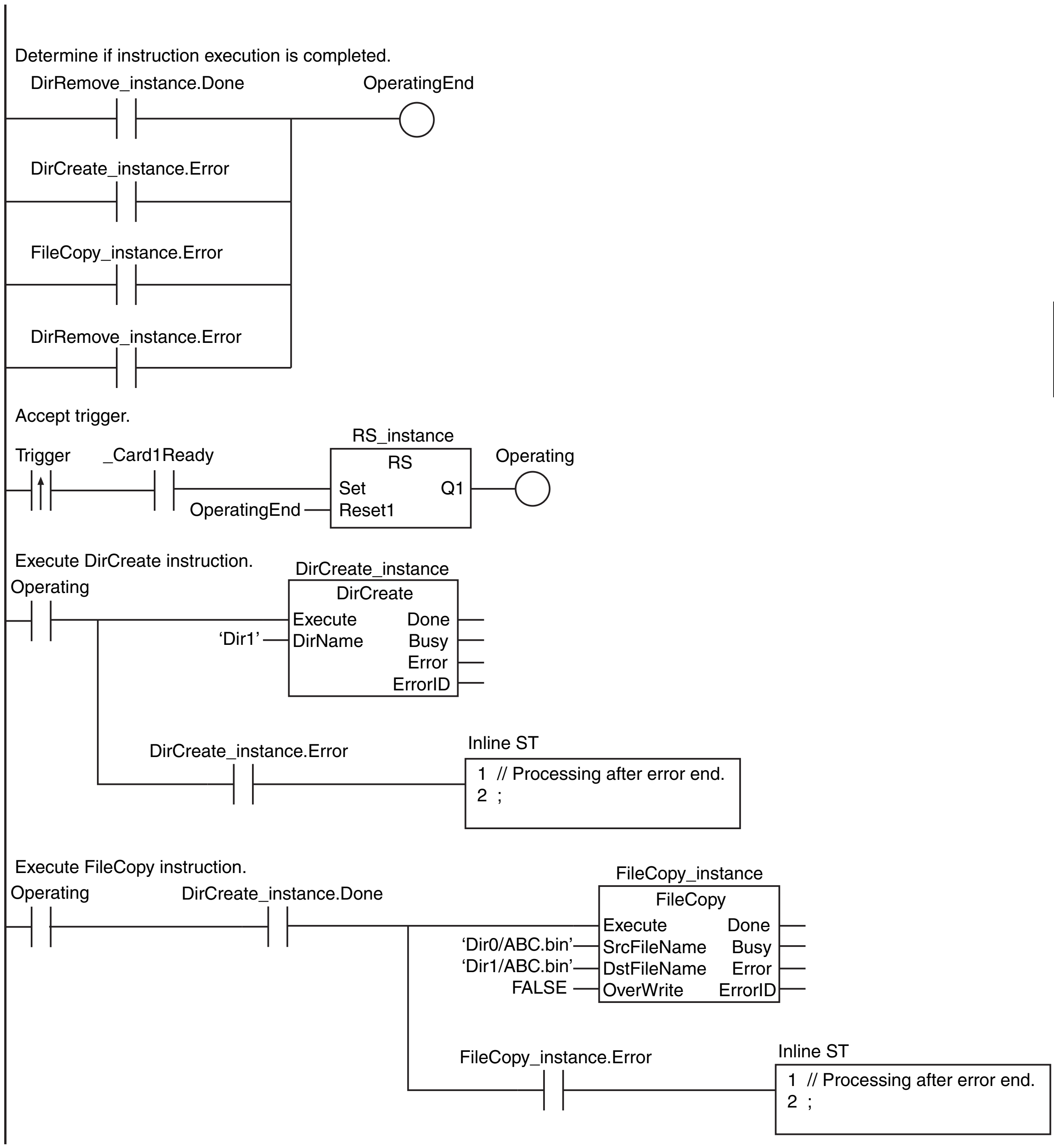
3 The DirRemove instruction is used to delete the directory ‘Dir0’ (the source of the copy).

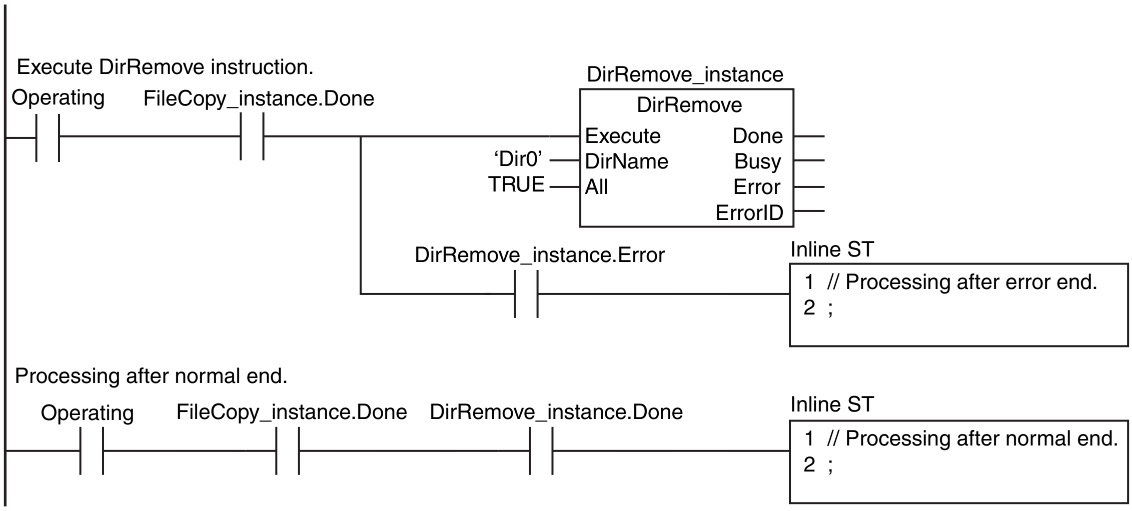


LD

Internal Variables	Variable	Data type	Initial value	Comment
	OperatingEnd	BOOL	FALSE	Processing completed.
	Trigger	BOOL	FALSE	Execution condition
	Operating	BOOL	FALSE	Processing
	RS_instance	RS		
	DirCreate_instance	DirCreate		
	FileCopy_instance	FileCopy		
	DirRemove_instance	DirRemove		

External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag


Determine if instruction execution is completed.
DirRemove_instance.Done
OperatingEnd
DirCreate_instance.Error
FileCopy_instance.Error
DirRemove_instance.Error
Accept trigger.
Trigger
_Card1Ready
RS_instance
RS
Set
Q1
Reset1
OperatingEnd
Operating
Execute DirCreate instruction.
Operating
DirCreate_instance
DirCreate
Execute
Done
'Dir1'
DirName
Busy
Error
ErrorID
DirCreate_instance.Error
Inline ST
1 // Processing after error end.
2 ;
Execute FileCopy instruction.
Operating
DirCreate_instance.Done
FileCopy_instance
FileCopy
Execute
Done
'Dir0/ABC.bin'
SrcFileName
Busy
'Dir1/ABC.bin'
DstFileName
Error
FALSE
OverWrite
ErrorID
FileCopy_instance.Error
Inline ST
1 // Processing after error end.
2 ;


Execute DirRemove instruction.
Operating
FileCopy_instance.Done
DirRemove_instance
DirRemove
Execute
Done
'Dir0'
DirName
Busy
TRUE
All
Error
ErrorID
DirRemove_instance.Error
Inline ST
1 // Processing after error end.
2 ;
Processing after normal end.
Operating
FileCopy_instance.Done
DirRemove_instance.Done
Inline ST
1 // Processing after normal end.
2 ;

ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	FALSE	Execution condition
	LastTrigger	BOOL	FALSE	Value of *Trigger* from previous task period
	OperatingStart	BOOL	FALSE	Processing started.
	Operating	BOOL	FALSE	Processing
	Stage	INT	0	Stage change
	DirCreate_instance	DirCreate		
	FileCopy_instance	FileCopy		
	DirRemove_instance	DirRemove		

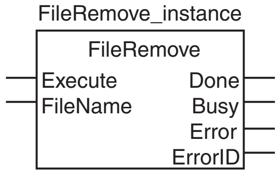
External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag

```
// Start sequence when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Card1Ready=TRUE) ) THEN
    OperatingStart:=TRUE;
    Operating     :=TRUE;
END_IF;
LastTrigger:=Trigger;

// Initialize instance.
IF (OperatingStart=TRUE) THEN
    DirCreate_instance(Execute:=FALSE);
    FileCopy_instance(Execute:=FALSE);
    DirRemove_instance(Execute:=FALSE);
    Stage         :=INT#1;
    OperatingStart:=FALSE;
END_IF;

// Execute instructions.
IF (Operating=TRUE) THEN
    CASE Stage OF
    1 :     // Create directory.
        DirCreate_instance(
            Execute:=TRUE,
            DirName:='Dir1');  // Directory name

        IF (DirCreate_instance.Done=TRUE) THEN
            Stage:=INT#2;      // Normal end
        END_IF;

        IF (DirCreate_instance.Error=TRUE) THEN
            Stage:=INT#99;     // Error end
        END_IF;

    2 :     // Copy file.
        FileCopy_instance(
            Execute    :=TRUE,
            SrcFileName:='Dir0/ABC.bin',  // Name of file to copy
            DstFileName:='Dir1/ABC.bin',  // Name of destination file
            OverWrite  :=FALSE);          // Prohibit overwrite.

        IF (FileCopy_instance.Done=TRUE) THEN
            Stage:=INT#3;
        END_IF;
```

```
        IF (FileCopy_instance.Error=TRUE) THEN
            Stage:=INT#99;
        END_IF;

    3 :     // Delete directory.
        DirRemove_instance(
            Execute :=TRUE,
            DirName :='Dir0',   // Directory name
            All     :=TRUE);    // Delete files and subdirectories.

        IF (DirRemove_instance.Done=TRUE) THEN
            Operating:=FALSE;   // Normal end
        END_IF;

        IF (DirRemove_instance.Error=TRUE) THEN
               Stage:=INT#99;      // Error end
        END_IF;

    99 :     // Processing after error end.
        Operating:=FALSE;
    END_CASE;
END_IF;
```

FileRemove

The FileRemove instruction deletes the specified file from the SD Memory Card.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FileRemove	Delete File	FB		FileRemove_instance(Execute, FileName, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
FileName	File name	Input	Name of file to delete	66 bytes max. (65 single-byte alphanumeric characters plus the final NULL character)	---	''

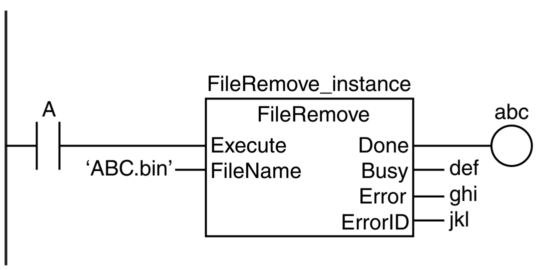
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileName																				OK

Function

The FileRemove instruction deletes the file specified by file name *FileName* from the SD Memory Card.

The following figure shows a programming example. Here, the file named ‘ABC.bin’ is deleted.

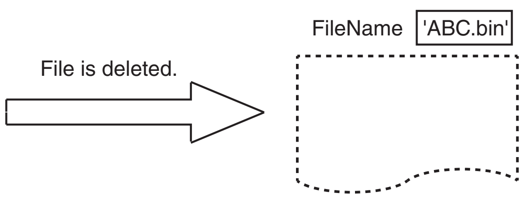
LD



ST

```
FileRemove_instance(A, ’ABC.bin’, abc,
                    def, ghi, jkl);
```

The FileRemove instruction deletes the file specified by *FileName* from the SD Memory Card.



Related System-defined Variables

Name	Meaning	Data type	Description
_Card1Ready	SD Memory Card Ready Flag	BOOL	This flag indicates if the SD Memory Card can be accessed by instructions and communications commands.*1 TRUE: Can be used. FALSE: Cannot be used.
_Card1Protect*2	SD Memory Card Write Protected Flag	BOOL	This flag indicates if the SD Memory Card is write protected when it is inserted and ready to use. TRUE: Write protected. FALSE: Not write protected.
_Card1Err*2	SD Memory Card Error Flag	BOOL	This flag indicates if an unspecified SD Memory Card (e.g., an SDHC card) is mounted or if the format is incorrect (i.e., not FAT16 or corrupted). TRUE: Error. FALSE: No error.
_Card1Access*2	SD Memory Card Access Flag	BOOL	This flag indicates if the SD Memory Card is currently being accessed. TRUE: Being accessed. FALSE: Not being accessed.
_Card1PowerFail	SD Memory Card Power Interruption Flag	BOOL	This flag indicates if an error occurred in completing processing when power was interrupted during access*3. This flag is not cleared automatically. TRUE: Error. FALSE: No error.

*1 For the NJ/NX-series, it is a precondition that the SD Memory Card is physically inserted and mounted normally. For an NY-series Controller, it is a precondition that the shared folder is detected by the Controller.

*2 These variables are not used for the NY-series Controller. They are fixed to FALSE.

*3 For the NJ/NX-series, this indicates an access to the SD Memory Card. For an NY-series Controller, this indicates an access to the shared folder.

Additional Information

The root directory of the file name is the top level of the SD Memory Card.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- If a file is open when the operating mode of the CPU Unit is changed to PROGRAM mode or when a major fault level Controller error occurs, the file is closed by the system. Any read/write operations that are in progress are completed to the end.
- For an NJ/NX-series CPU Unit, if a file is open when the power supply is stopped with the SD Memory Card power supply switch, the file is not corrupted.
- For an NJ/NX-series CPU Unit, if a file is open and the SD Memory Card is removed before the SD Memory Card power supply switch is pressed, the contents of the file will sometimes be corrupted. Always turn OFF the power supply before removing the SD Memory Card.

- For an NJ/NX-series CPU Unit, if a file is open when the power supply is stopped or the SD Memory Card is removed, it will not be possible to read or write the file even if the SD Memory Card is inserted again.
- Do not simultaneously access the same file. Perform exclusive control of SD Memory Card instructions in the user program.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The SD Memory Card is not in a usable condition.
  - The SD Memory Card is write protected.
  - The file specified by *FileName* does not exist.
  - The file specified by *FileName* is being accessed.
  - A file with the name *FileName* already exits and the file is write protected.
  - If more than four SD Memory Card instructions that do not have a *FileID* variable (i.e., FileWriteVar, FileReadVar, FileCopy, DirCreate, FileRemove, DirRemove, and FileRename) are executed at the same time.
  - The value of *FileName* exceeds the maximum number of characters allowed in a file name.
  - An error that prevents access occurs during SD Memory Card access.

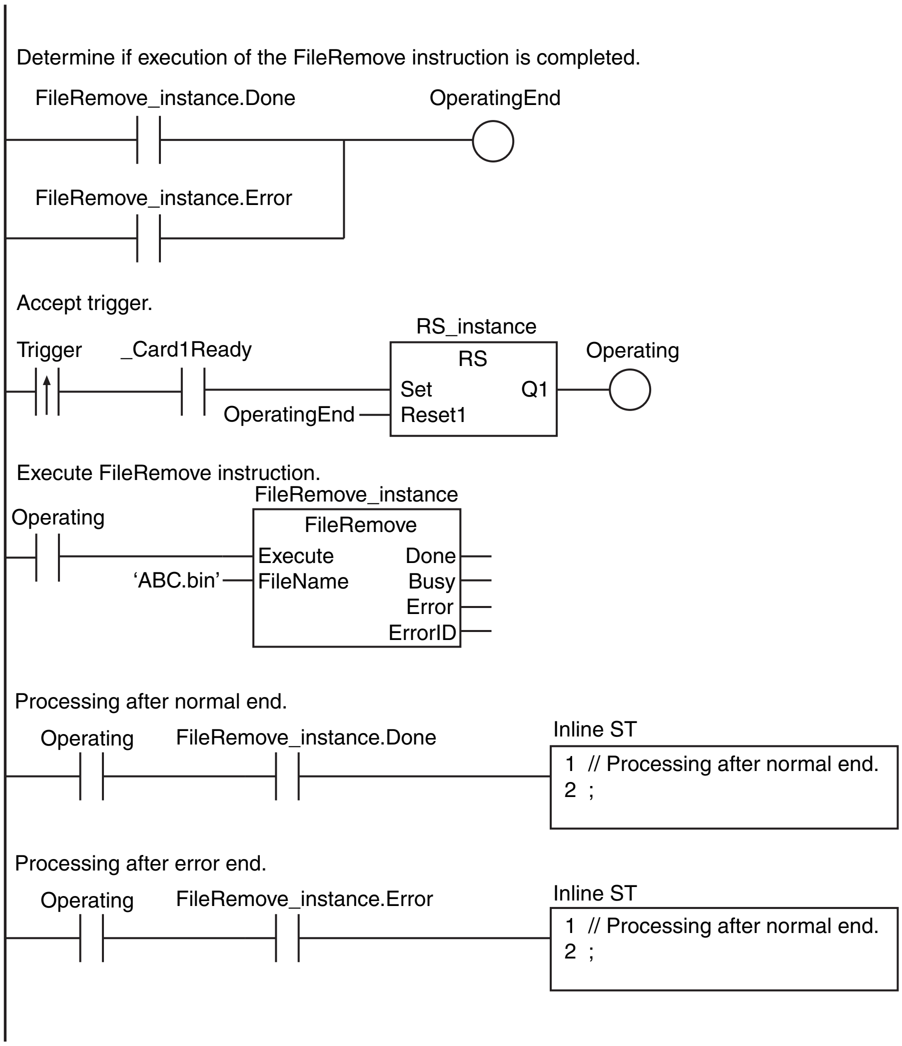
Sample Programming

In this sample, the file named 'ABC.bin' is deleted from the SD Memory Card.

LD

Internal Variables	Variable	Data type	Initial value	Comment
	OperatingEnd	BOOL	FALSE	Processing completed.
	Trigger	BOOL	FALSE	Execution condition
	Operating	BOOL	FALSE	Processing
	RS_instance	RS		
	FileRemove_instance	FileRemove		

External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag


Determine if execution of the FileRemove instruction is completed.
FileRemove_instance.Done
OperatingEnd
FileRemove_instance.Error
Accept trigger.
RS_instance
Trigger
_Card1Ready
RS
Operating
Set
Q1
OperatingEnd
Reset1
Execute FileRemove instruction.
FileRemove_instance
Operating
FileRemove
Execute
Done
'ABC.bin'
FileName
Busy
Error
ErrorID
Processing after normal end.
Inline ST
Operating
FileRemove_instance.Done
1 // Processing after normal end.
2 ;
Processing after error end.
Inline ST
Operating
FileRemove_instance.Error
1 // Processing after normal end.
2 ;

ST

Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	FALSE	Execution condition
	LastTrigger	BOOL	FALSE	Value of *Trigger* from previous task period
	OperatingStart	BOOL	FALSE	Processing started.
	Operating	BOOL	FALSE	Processing
	FileRemove_instance	FileRemove		

External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag

```
// Start sequence when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Card1Ready=TRUE) ) THEN
    OperatingStart:=TRUE;
    Operating     :=TRUE;
END_IF;
LastTrigger:=Trigger;

// Initialize instance.
IF (OperatingStart=TRUE) THEN
    FileRemove_instance(Execute:=FALSE);
    OperatingStart:=FALSE;
END_IF;

// Execute FileRemove instruction.
IF (Operating=TRUE) THEN
    FileRemove_instance(
            Execute :=TRUE,
            FileName:='ABC.bin');  // File name

    IF (FileRemove_instance.Done=TRUE) THEN
        Operating:=FALSE;           // Normal end
    END_IF;

    IF (FileRemove_instance.Error=TRUE) THEN
        Operating:=FALSE;           // Error end
    END_IF;
END_IF;
```

FileRename

The FileRename instruction changes the name of the specified file or directory in the SD Memory Card.

Instruction	Name	FB/FUN	Graphic expression	ST expression
FileRename	Change File Name	FB	FileRename_instance FileRename Execute Done FileName Busy NewName Error OverWrite ErrorID	FileRename_instance(Execute, FileName, NewName, OverWrite, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
FileName	Original file name	Input	Original file name	66 bytes max. (65 single-byte alphanumeric characters plus the final NULL character)	---	''
NewName	New file name		New file name			
OverWrite	Overwrite enable		TRUE: Enable overwrite. FALSE: Prohibit overwrite.	Depends on data type.		FALSE

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
FileName																				OK
NewName																				OK
OverWrite	OK																			

Function

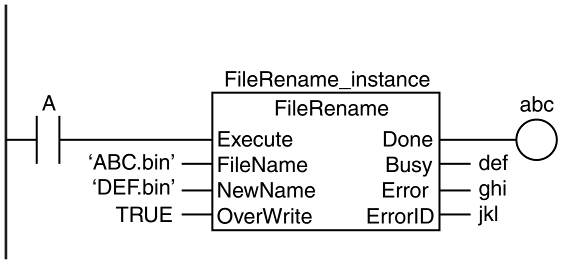
The FileRename instruction changes the name of the file or directory specified by original file name *FileName* to new file name *NewName* in the SD Memory Card.

If a file or directory with the name *NewName* already exists in the SD Memory Card, the following processing is performed depending on the value of overwrite enable *OverWrite*.

Value of *OverWrite*	Treatment
TRUE (Enable overwrite.)	The existing file or directory is overwritten.
FALSE (Prohibit overwrite.)	The file or directory is not overwritten and an error occurs.

The following figure shows a programming example. Here, the name of the file 'ABC.bin' is changed to 'DEF.bin.'

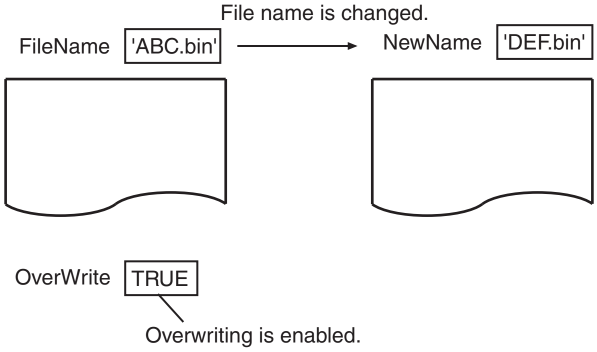
LD



ST

```
FileRename_instance(A, 'ABC.bin', 'DEF.bin',
                    TRUE, abc, def, ghi, jkl);
```

The FileRename instruction changes the name of the file specified by original file name *FileName* to new file name *NewName* in the SD Memory Card. If the file already exists, it is overwritten.



Related System-defined Variables

Name	Meaning	Data type	Description
_Card1Ready	SD Memory Card Ready Flag	BOOL	This flag indicates if the SD Memory Card can be accessed by instructions and communications commands.*1 TRUE: Can be used. FALSE: Cannot be used.
_Card1Protect*2	SD Memory Card Write Protected Flag	BOOL	This flag indicates if the SD Memory Card is write protected when it is inserted and ready to use. TRUE: Write protected. FALSE: Not write protected.
_Card1Err*2	SD Memory Card Error Flag	BOOL	This flag indicates if an unspecified SD Memory Card (e.g., an SDHC card) is mounted or if the format is incorrect (i.e., not FAT16 or corrupted). TRUE: Error. FALSE: No error.
_Card1Access*2	SD Memory Card Access Flag	BOOL	This flag indicates if the SD Memory Card is currently being accessed. TRUE: Being accessed. FALSE: Not being accessed.
_Card1PowerFail	SD Memory Card Power Interruption Flag	BOOL	This flag indicates if an error occurred in completing processing when power was interrupted during access*3. This flag is not cleared automatically. TRUE: Error. FALSE: No error.

*1 For the NJ/NX-series, it is a precondition that the SD Memory Card is physically inserted and mounted normally. For an NY-series Controller, it is a precondition that the shared folder is detected by the Controller.

*2 These variables are not used for the NY-series Controller. They are fixed to FALSE.

*3 For the NJ/NX-series, this indicates an access to the SD Memory Card. For an NY-series Controller, this indicates an access to the shared folder.

Additional Information

The root directory of the file name is the top level of the SD Memory Card.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- If the directories are different for *FileName* and *NewName*, the file is moved to the directory that is specified with *NewName*.
- If a file is open when the operating mode of the CPU Unit is changed to PROGRAM mode or when a major fault level Controller error occurs, the file is closed by the system. Any read/write operations that are in progress are completed to the end.
- For an NJ/NX-series CPU Unit, if a file is open when the power supply is stopped with the SD Memory Card power supply switch, the file is not corrupted.
- For an NJ/NX-series CPU Unit, if a file is open and the SD Memory Card is removed before the SD Memory Card power supply switch is pressed, the contents of the file will sometimes be corrupted. Always turn OFF the power supply before removing the SD Memory Card.
- For an NJ/NX-series CPU Unit, if a file is open when the power supply is stopped or the SD Memory Card is removed, it will not be possible to read or write the file even if the SD Memory Card is inserted again.
- Do not simultaneously access the same file. Perform exclusive control of SD Memory Card instructions in the user program.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The SD Memory Card is not in a usable condition.
  - The SD Memory Card is write protected.
  - The file directory specified with *FileName* does not exist.
  - The value of *FileName* or *NewName* is not a valid file name or directory name.
  - The file specified by *FileName* is being accessed.
  - There is a subdirectory in the directory that was specified for *FileName* and the value of *OverWrite* is TRUE.
  - A file with the name *NewName* already exits and the value of *OverWrite* is FALSE.
  - A file with the name *NewName* already exits, the file is write protected, and the value of *OverWrite* is TRUE.
  - If more than four SD Memory Card instructions that do not have a *FileID* variable (i.e., FileWriteVar, FileReadVar, FileCopy, DirCreate, FileRemove, DirRemove, and FileRename) are executed at the same time.
  - The value of *NewName* exceeds the maximum number of characters allowed in a file name or directory name.
  - An error that prevents access occurs during SD Memory Card access.
  - The maximum number of directories is exceeded.

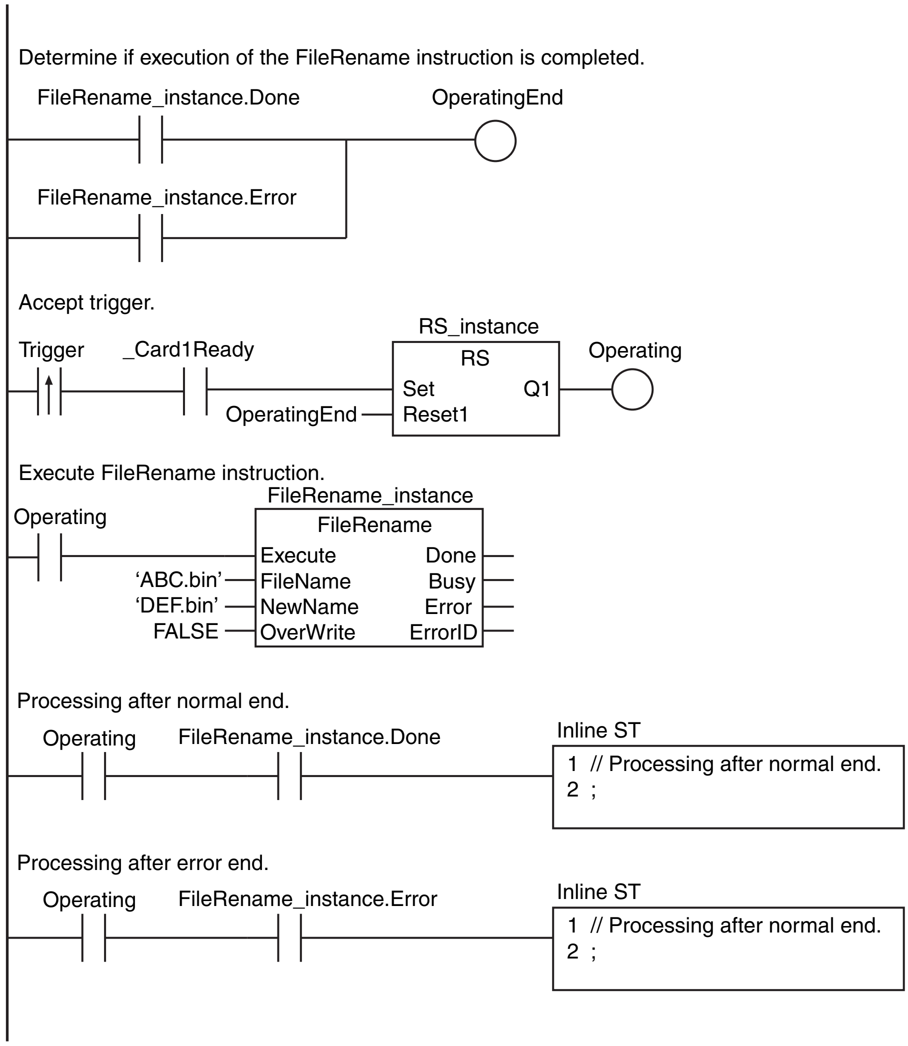
Sample Programming

In this sample, the name of the file 'ABC.bin' is changed to 'DEF.bin' on the SD Memory Card.

LD

Internal Variables	Variable	Data type	Initial value	Comment
	OperatingEnd	BOOL	FALSE	Processing completed.
	Trigger	BOOL	FALSE	Execution condition
	Operating	BOOL	FALSE	Processing
	RS_instance	RS		
	FileRename_instance	FileRename		

External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag



ST

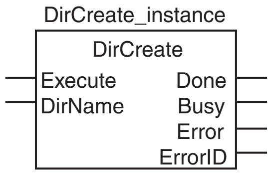
Internal Variables	Variable	Data type	Initial value	Comment
	Trigger	BOOL	FALSE	Execution condition
	LastTrigger	BOOL	FALSE	Value of *Trigger* from previous task period
	OperatingStart	BOOL	FALSE	Processing started.
	Operating	BOOL	FALSE	Processing
	FileRename_instance	FileRename		

External Variables	Variable	Data type	Comment
	_Card1Ready	BOOL	SD Memory Card Ready Flag

```
// Start sequence when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) AND (_Card1Ready=TRUE) ) THEN
    OperatingStart:=TRUE;
    Operating     :=TRUE;
END_IF;
LastTrigger:=Trigger;

// Initialize instance.
IF (OperatingStart=TRUE) THEN
    FileRename_instance(Execute:=FALSE);
    OperatingStart:=FALSE;
END_IF;

// Execute FileRename instruction.
IF (Operating=TRUE) THEN
    FileRename_instance(
            Execute  :=TRUE,
            FileName :='ABC.bin',  // Original file name
            NewName  :='DEF.bin',  // New file name
            OverWrite:=FALSE);     // Prohibit overwrite.

    IF (FileRename_instance.Done=TRUE) THEN
        Operating:=FALSE;           // Normal end
    END_IF;

    IF (FileRename_instance.Error=TRUE) THEN
        Operating:=FALSE;           // Error end
    END_IF;
END_IF;
```

DirCreate

The DirCreate instruction creates a directory with the specified name in the SD Memory Card.

Instruction	Name	FB/FUN	Graphic expression	ST expression
DirCreate	Create Directory	FB		DirCreate_instance(Execute, DirName, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
DirName	Directory to create	Input	Name of directory to create	66 bytes max. (65 single-byte alphanumeric characters plus the final NULL character)	---	''

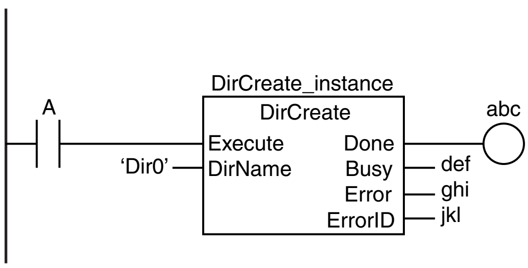
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
DirName																				OK

Function

The DirCreate instruction creates a directory with the name specified by directory to create *Dir* in the SD Memory Card.

The following figure shows a programming example. Here, a directory named ‘Dir0’ is created.

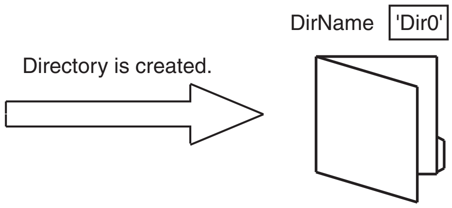
LD



ST

```
DirCreate_instance(A, ’Dir0’, abc,
                   def, ghi, jkl);
```

The DirCreate instruction creates a directory with the name specified by *DirName* in the SD Memory Card.



Related System-defined Variables

Name	Meaning	Data type	Description
_Card1Ready	SD Memory Card Ready Flag	BOOL	This flag indicates if the SD Memory Card can be accessed by instructions and communications commands.*1 TRUE: Can be used. FALSE: Cannot be used.
_Card1Protect*2	SD Memory Card Write Protected Flag	BOOL	This flag indicates if the SD Memory Card is write protected when it is inserted and ready to use. TRUE: Write protected. FALSE: Not write protected.
_Card1Err*2	SD Memory Card Error Flag	BOOL	This flag indicates if an unspecified SD Memory Card (e.g., an SDHC card) is mounted or if the format is incorrect (i.e., not FAT16 or corrupted). TRUE: Error. FALSE: No error.
_Card1Access*2	SD Memory Card Access Flag	BOOL	This flag indicates if the SD Memory Card is currently being accessed. TRUE: Being accessed. FALSE: Not being accessed.
_Card1PowerFail	SD Memory Card Power Interruption Flag	BOOL	This flag indicates if an error occurred in completing processing when power was interrupted during access*3. This flag is not cleared automatically. TRUE: Error. FALSE: No error.

*1 For the NJ/NX-series, it is a precondition that the SD Memory Card is physically inserted and mounted normally. For an NY-series Controller, it is a precondition that the shared folder is detected by the Controller.

*2 These variables are not used for the NY-series Controller. They are fixed to FALSE.

*3 For the NJ/NX-series, this indicates an access to the SD Memory Card. For an NY-series Controller, this indicates an access to the shared folder.

Additional Information

The root directory of the file name is the top level of the SD Memory Card.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.

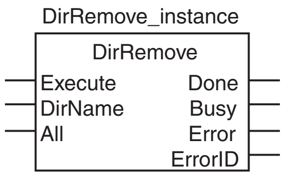
- If a file is open when the operating mode of the CPU Unit is changed to PROGRAM mode or when a major fault level Controller error occurs, the file is closed by the system. Any read/write operations that are in progress are completed to the end.
- For an NJ/NX-series CPU Unit, if a file is open when the power supply is stopped with the SD Memory Card power supply switch, the file is not corrupted.
- For an NJ/NX-series CPU Unit, if a file is open and the SD Memory Card is removed before the SD Memory Card power supply switch is pressed, the contents of the file will sometimes be corrupted. For an NJ/NX-series CPU Unit, if a file is open and the SD Memory Card is removed before the SD Memory Card power supply switch is pressed, the contents of the file will sometimes be corrupted. Always turn OFF the power supply before removing the SD Memory Card.
- For an NJ/NX-series CPU Unit, if a file is open when the power supply is stopped or the SD Memory Card is removed, it will not be possible to read or write the file even if the SD Memory Card is inserted again.
- Do not simultaneously access the same file. Perform exclusive control of SD Memory Card instructions in the user program.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The SD Memory Card is not in a usable condition.
  - The SD Memory Card is write protected.
  - There is insufficient space available on the SD Memory Card.
  - The maximum number of directories is exceeded.
  - The directory specified by *DirName* already exists.
  - If more than four SD Memory Card instructions that do not have a *FileID* variable (i.e., FileWriteVar, FileReadVar, FileCopy, DirCreate, FileRemove, DirRemove, and FileRename) are executed at the same time.
  - The value of *DirName* is not a valid directory name.
  - The value of *DirName* exceeds the maximum number of characters allowed in a directory name.
  - An error that prevents access occurs during SD Memory Card access.
  - The file specified by *FileName* is being accessed.

Sample Programming

Refer to the sample programming that is provided for the FileCopy instruction (page 2-1310).

DirRemove

The DirRemove instruction deletes the specified directory from the SD Memory Card.

Instruction	Name	FB/FUN	Graphic expression	ST expression
DirRemove	Delete Directory	FB		DirRemove_instance(Execute, DirName, All, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
DirName	Directory to delete	Input	Directory to delete	66 bytes max. (65 single-byte alphanumeric characters plus the final NULL character)	---	''
All	All designation		Specifies whether to delete files and subdirectories inside specified directory TRUE: Delete files and subdirectories. FALSE: Do not delete.	Depends on data type.		FALSE

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
DirName																				OK
All	OK																			

Function

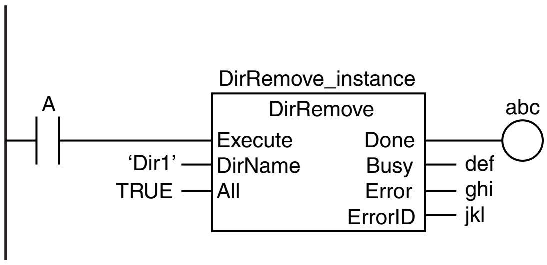
The DirRemove instruction deletes the directory with the name specified by directory to delete *Dir* from the SD Memory Card.

If there are files or subdirectories in the specified directory, the following processing is performed according to the value of all designation *All*.

Value of *All*	Treatment
TRUE	All files and subdirectories are deleted along with the specified directory.
FALSE	The specified directory is not deleted and an error occurs.

The following figure shows a programming example. Here, a directory named ‘Dir1’ is deleted.

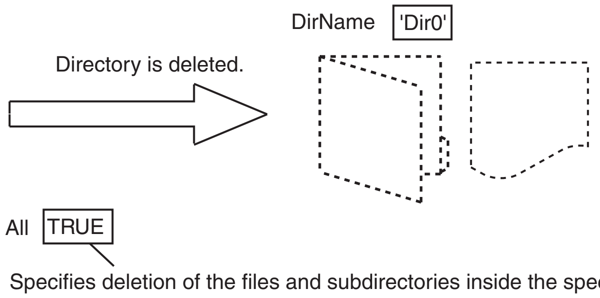
LD



ST

```
DirRemove_instance(A, ’Dir1’, TRUE, abc,
                def, ghi, jkl);
```

The DirRemove instruction deletes the directory with the name specified by *DirName* from the SD Memory Card. Files and subdirectories inside specified directory are deleted too.



Related System-defined Variables

Name	Meaning	Data type	Description
_Card1Ready	SD Memory Card Ready Flag	BOOL	This flag indicates if the SD Memory Card can be accessed by instructions and communications commands.*1 TRUE: Can be used. FALSE: Cannot be used.
_Card1Protect*2	SD Memory Card Write Protected Flag	BOOL	This flag indicates if the SD Memory Card is write protected when it is inserted and ready to use. TRUE: Write protected. FALSE: Not write protected.
_Card1Err*2	SD Memory Card Error Flag	BOOL	This flag indicates if an unspecified SD Memory Card (e.g., an SDHC card) is mounted or if the format is incorrect (i.e., not FAT16 or corrupted). TRUE: Error. FALSE: No error.
_Card1Access*2	SD Memory Card Access Flag	BOOL	This flag indicates if the SD Memory Card is currently being accessed. TRUE: Being accessed. FALSE: Not being accessed.
_Card1PowerFail	SD Memory Card Power Interruption Flag	BOOL	This flag indicates if an error occurred in completing processing when power was interrupted during access*3. This flag is not cleared automatically. TRUE: Error. FALSE: No error.

*1 For the NJ/NX-series, it is a precondition that the SD Memory Card is physically inserted and mounted normally. For an NY-series Controller, it is a precondition that the shared folder is detected by the Controller.

*2 These variables are not used for the NY-series Controller. They are fixed to FALSE.

*3 For the NJ/NX-series, this indicates an access to the SD Memory Card. For an NY-series Controller, this indicates an access to the shared folder.

Additional Information

The root directory of the file name is the top level of the SD Memory Card.

Precautions for Correct Use

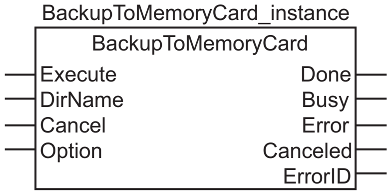
- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* on page 2-3 for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- If a file is open when the operating mode of the CPU Unit is changed to PROGRAM mode or when a major fault level Controller error occurs, the file is closed by the system. Any read/write operations that are in progress are completed to the end.
- For an NJ/NX-series CPU Unit, if a file is open when the power supply is stopped with the SD Memory Card power supply switch, the file is not corrupted.
- For an NJ/NX-series CPU Unit, if a file is open and the SD Memory Card is removed before the SD Memory Card power supply switch is pressed, the contents of the file will sometimes be corrupted. Always turn OFF the power supply before removing the SD Memory Card.
- For an NJ/NX-series CPU Unit, if a file is open when the power supply is stopped or the SD Memory Card is removed, it will not be possible to read or write the file even if the SD Memory Card is inserted again.
- If the directory that is specified with *DirName* is write protected, an error occurs and the directory is not deleted. However, any files or directories that are not write-protected inside that directory are deleted.
- Do not simultaneously access the same file. Perform exclusive control of SD Memory Card instructions in the user program.
- An error occurs in the following cases. *Error* will change to TRUE.
  - The SD Memory Card is not in a usable condition.
  - The SD Memory Card is write protected.
  - If the value of *All* is TRUE and the directory specified with *DirName* is being accessed by another instruction.
  - If the value of *All* is FALSE and the directory specified with *DirName* contains a file or directory.
  - The directory specified by *DirName* is write-protected.
  - The directory that is specified with *DirName* contains write-protected files or write-protected directories.
  - If more than four SD Memory Card instructions that do not have a *FileID* variable (i.e., FileWriteVar, FileReadVar, FileCopy, DirCreate, FileRemove, DirRemove, and FileRename) are executed at the same time.
  - The directory specified by *DirName* does not exist.
  - The value of *DirName* exceeds the maximum number of characters allowed in a directory name.
  - An error that prevents access occurs during SD Memory Card access.

Sample Programming

Refer to the sample programming that is provided for the FileCopy instruction (page 2-1310).

BackupToMemoryCard

The BackupToMemoryCard instruction backs up data to an SD Memory Card.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
BackupTo MemoryCard	SD Memory Card Backup	FB		BackupToMemoryCard _instance(Execute, DirName, Cancel, Option, Done, Busy, Error, Canceled, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
DirName	Directory to save in	Input	Name of directory in which to save the backup data	64 bytes max. (63 single-byte alphanumeric characters plus the final NULL character)	---	''
Cancel	Cancel	Input	Canceling the backup TRUE: Cancel FALSE: Do not cancel	Depends on data type.	---	FALSE
Option	For future expansion	Input	This variable is for future expansion. It is not necessary to connect a parameter.	---	---	---
Canceled	Cancel completed	Output	A flag that indicates if canceling was completed TRUE: Canceling completed FALSE: Canceling failed	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
DirName																				OK
Cancel	OK																			
Option	For future expansion. It is not necessary to connect a parameter.																			
Canceled	OK																			

Function

The BackupToMemoryCard instruction backs up data to an SD Memory Card.

This instruction performs the same processing as the processing that is performed for the front panel switch on the CPU Unit, the *_Card1BkupCmd* system-defined variable, or the SD Memory Card backup performed from the SD Memory Card Window on the Sysmac Studio.

Specify the name of the directory in which to save the backup data with *DirName*.

If the value of *DirName* is “ ” (i.e., a text string with a length of 0 characters), the backup data is saved in the root directory of the SD Memory Card.

DirName can be omitted. If you omit *DirName*, the following data is save in the following directory.

Instruction execution	Directory to save in
1st execution	Root directory
2nd execution and beyond	The previously specified directory

If the directory that is specified with *DirName* does not exist in the SD Memory Card, a new directory is created and the backup data is saved in it.

If a file with the same name as the backup file already exists in the directory specified with *DirName*, the backup file is overwritten.

If the value of *Cancel* changes to TRUE during backup processing, the backup processing is canceled. If backup processing is canceled, the backup file will not be created. If a backup file already exists in the directory specified with *DirName*, the backup file is not overwritten and remains unchanged.

You can cancel only the backup processing that is being executed for the same function block instance.

When canceling is completed, the value of *Canceled* changes to TRUE. Depending on when the value of *Cancel* changes to TRUE, it might be too late to cancel processing, and backup processing may be competed to the end. If canceling was not performed in time, the value of *Canceled* will be FALSE and the value of *Done* will be TRUE.

If the value of *Cancel* is TRUE, backup processing is not performed even if the value of *Execute* is TRUE.

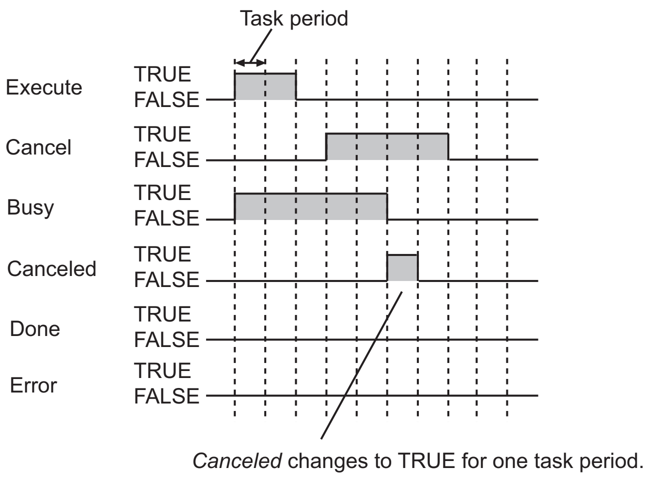
Option is for future expansion. Do not connect a parameter to it.

Timing Chart for Canceling

Timing charts for the instruction variables are provided below for canceling.

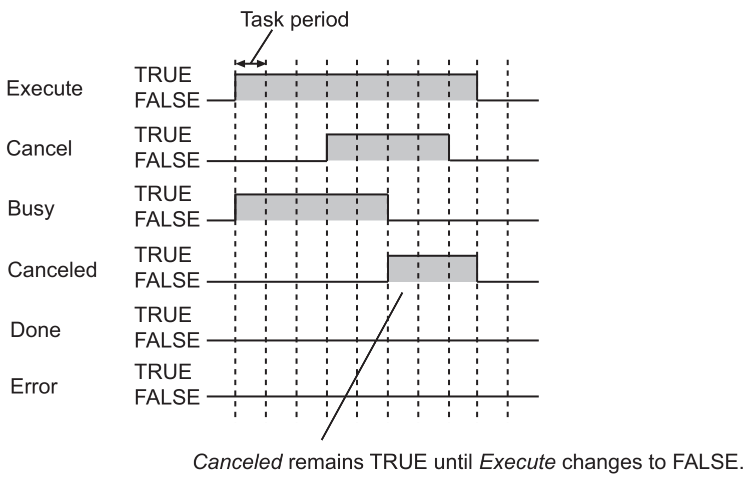
● When Canceling Is Successful, and *Execute* Changes to FALSE Before *Canceled* Changes to TRUE

- Backup processing is executed when the value of *Execute* changes to TRUE. The value of *Busy* changes to TRUE.
- Backup processing is canceled when the value of *Cancel* changes to TRUE.
- When canceling is completed, the value of *Busy* changes to FALSE and the value of *Canceled* changes to TRUE.
- The value of *Execute* is changed to FALSE before the value of *Canceled* changes to TRUE.
- The value of *Canceled* changes to FALSE after one task period.
- Because canceling was successful, the value of *Done* changes to FALSE.



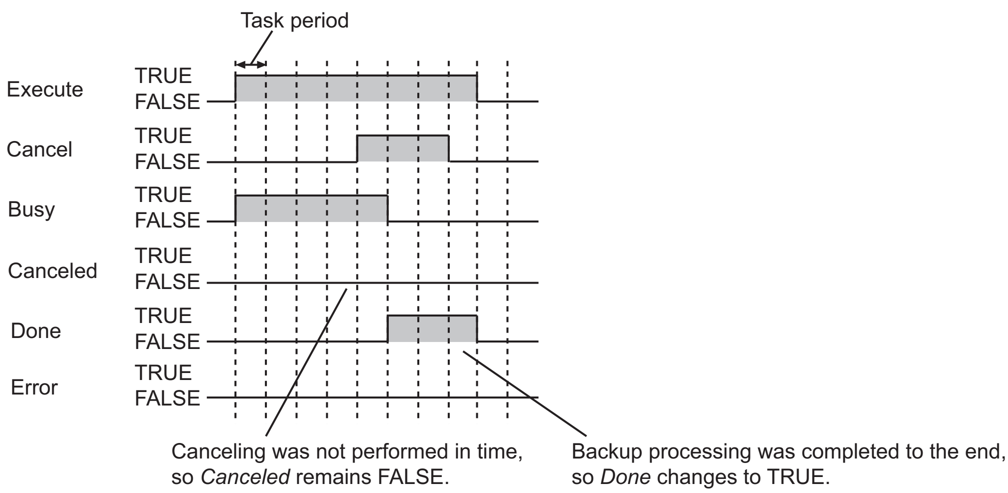
When Canceling Is Successful, and *Execute* Changes to FALSE After *Canceled* Changes to TRUE

- Backup processing is executed when the value of *Execute* changes to TRUE. The value of *Busy* changes to TRUE.
- Backup processing is canceled when the value of *Cancel* changes to TRUE.
- When canceling is completed, the value of *Busy* changes to FALSE and the value of *Canceled* changes to TRUE.
- The value of *Execute* is changed to FALSE after the value of *Canceled* changes to TRUE.
- The value of *Canceled* remains TRUE until the value of *Execute* changes to FALSE.
- Because canceling was successful, the value of *Done* changes to FALSE.



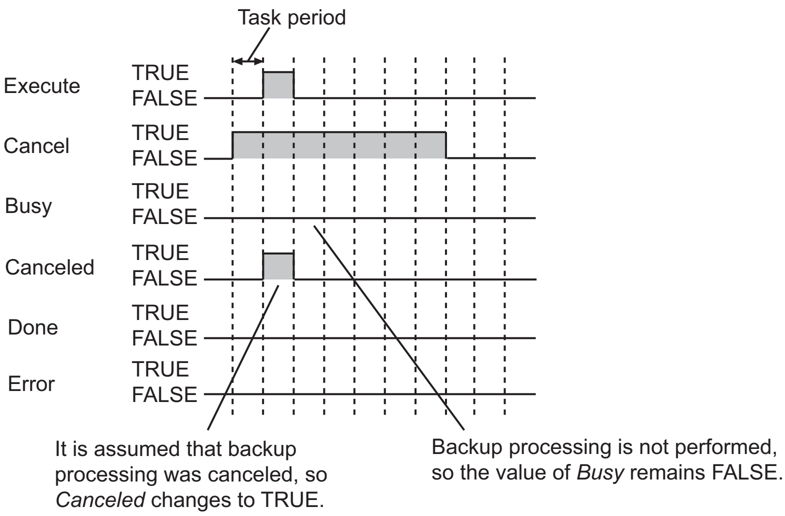
When Canceling Is Not Performed in Time

- Backup processing is executed when the value of *Execute* changes to TRUE. The value of *Busy* changes to TRUE.
- The value of *Cancel* is changed to TRUE. Backup processing continues because canceling was not performed in time.
- When backup processing is completed, the value of *Busy* changes to FALSE.
- Backup processing was completed to the end, so the value of *Done* changes to TRUE.
- Canceling was not performed in time, so the value of *Canceled* remains FALSE.



When the Value of *Execute* Is Changed to TRUE While the Value of *Cancel* Is TRUE

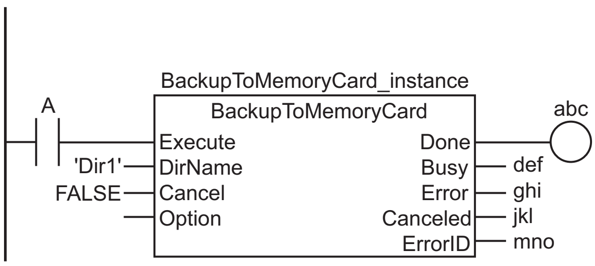
- The value of *Cancel* is changed to TRUE.
- Backup processing is not executed even if the value of *Execute* is changed to TRUE. Therefore, the value of *Busy* remains FALSE.
- It is assumed that backup processing was canceled, so the value of *Canceled* changes to TRUE.
- When the value of *Execute* is changed to FALSE, the value of *Canceled* changes to FALSE.



Notation Example

The following figure shows a programming example. The backup file is saved in a directory called Dir1.

LD



ST

```
BackupToMemoryCard_instance(A, 'Dir1', FALSE,
                            , abc, def, ghi, jkl,
                            mno);
```

Related System-defined Variables

Variable	Name	Data type	Description
_Card1Ready	SD Memory Card Ready Flag	BOOL	This flag indicates if the SD Memory Card can be accessed by instructions and communications commands.*1 TRUE: Can be used. FALSE: Cannot be used.
_Card1Protect*2	SD Memory Card Write Protected Flag	BOOL	This flag indicates if the SD Memory Card is write protected when it is inserted and ready to use. TRUE: Write protected. FALSE: Not write protected.
_Card1Err*2	SD Memory Card Error Flag	BOOL	This flag indicates if an unspecified SD Memory Card (e.g., an SDHC card) is mounted or if the format is incorrect (i.e., not FAT16 or corrupted). TRUE: Error. FALSE: No error.
_Card1Access*2	SD Memory Card Access Flag	BOOL	This flag indicates if the SD Memory Card is currently being accessed. TRUE: Being accessed. FALSE: Not being accessed.
_Card1Deteriorated*2	SD Memory Card Life Warning Flag	BOOL	This flag indicates if the end of the life of the SD Memory Card is detected. TRUE: End of life detected. FALSE: Not detected.
_Card1PowerFail	SD Memory Card Power Interruption Flag	BOOL	This flag indicates if an error occurred in completing processing when power was interrupted during access*3. This flag is not cleared automatically. TRUE: Error. FALSE: No error.
_BackupBusy	Backup Function Busy Flag	BOOL	This flag indicates if a backup, restoration, or verification is in progress. TRUE: Backup, restore, or compare operation is in progress. FALSE: Backup, restore, or compare operation is not in progress.

*1 For the NJ/NX-series, it is a precondition that the SD Memory Card is physically inserted and mounted normally. For an NY-series Controller, it is a precondition that the shared folder is detected by the Controller.

*2 These variables are not used for the NY-series Controller. They are fixed to FALSE.

*3 For the NJ/NX-series, this indicates an access to the SD Memory Card. For an NY-series Controller, this indicates an access to the shared folder.

Additional Information

- Refer to the *NJ/NX-series CPU Unit Software User's Manual* (Cat. No. W501) or *NY-series Industrial Panel PC / Industrial Box PC Software User's Manual* (Cat. No. W558) for details on the backup functions.
- The root directory of the file name is the top level of the SD Memory Card.

Precautions for Correct Use

- Execution of this instruction is continued until processing is completed even if the value of *Execute* changes to FALSE or the execution time exceeds the task period. The value of *Done* changes to TRUE when processing is completed. Use this to confirm normal completion of processing.
- Refer to *Using this Section* (page 2-3) for a timing chart for *Execute*, *Done*, *Busy*, and *Error*.
- For an NJ/NX-series CPU Unit, if a file is open and the SD Memory Card is removed before the SD Memory Card power supply switch is pressed, the contents of the file will sometimes be corrupted. Always turn OFF the power supply before removing the SD Memory Card.
- Even if data backup to the SD Memory Card is prohibited, you can execute this instruction to backup the data. No error will occur.
- The values of the following system-defined variables that are related to backup do not change when this instruction is executed.
  - SD Memory Card Backup Command: _CardBkupCmd
  - SD Memory Card Backup Status: _Card1BkupSta
- Do not read or write backup-related files during execution of this instruction. If you read a file that is being written, unexpected processing may occur.
- Backup processing will continue even if the operating mode of the CPU Unit is changed during execution of this instruction. If you change the operating mode from RUN mode to PROGRAM mode and then back to RUN mode, the value of *Busy* will be FALSE even if backup processing is in progress. If you cancel backup processing under that condition, the value of *Canceled* will change to TRUE.
- An error will occur in the following cases. *Error* will change to TRUE.
  - The SD Memory Card is not in a usable condition.
  - The SD Memory Card is write protected.
  - There is insufficient space available on the SD Memory Card.
  - The maximum number of files or directories is exceeded.
  - A file already exists with the same name as the name specified with *DirName*.
  - The value of *DirName* is not a valid directory name.
  - An error that prevents access occurs during SD Memory Card access.
  - Another backup operation is already in progress.
  - Backup processing failed.

Version Information

A CPU Unit with unit version 1.08 or later and Sysmac Studio version 1.09 or higher are required to use this instruction.

Sample Programming

In this example, the BackupToMemoryCard instruction backs up data to an SD Memory Card every day just after midnight. The backup-related files are stored in directories named /Backup/yyyy-mm-dd in the SD Memory Card. The directory name gives the date when the backup was executed. “yyyy” is the year, “mm” is the month, and “dd” is the day of the month.

Touch Panel Specifications

This example assumes that a touch panel is connected to the Controller.

The touch panel has the following lamps.

Lamp name	Description
Backup normal end lamp	Lights when backup processing ends normally.
Backup canceled lamp	Lights when backup processing is successfully canceled.
Backup error end lamp	Lights when backup processing ends in an error.
SD Memory Card life warning lamp	Lights when the life of the SD Memory Card was exceeded.
SD Memory Card power interrupted lamp	Lights when power to the SD Memory Card was interrupted during backup processing.

The touch panel also has the following buttons.

Button name	Operation when button is pressed
Lamps OFF button	Turns OFF the Backup Normal End Lamp, Backup Canceled Lamp, Backup Error End Lamp, and SD Memory Card Power Interrupted Lamp.
Cancel button	Cancels the backup.

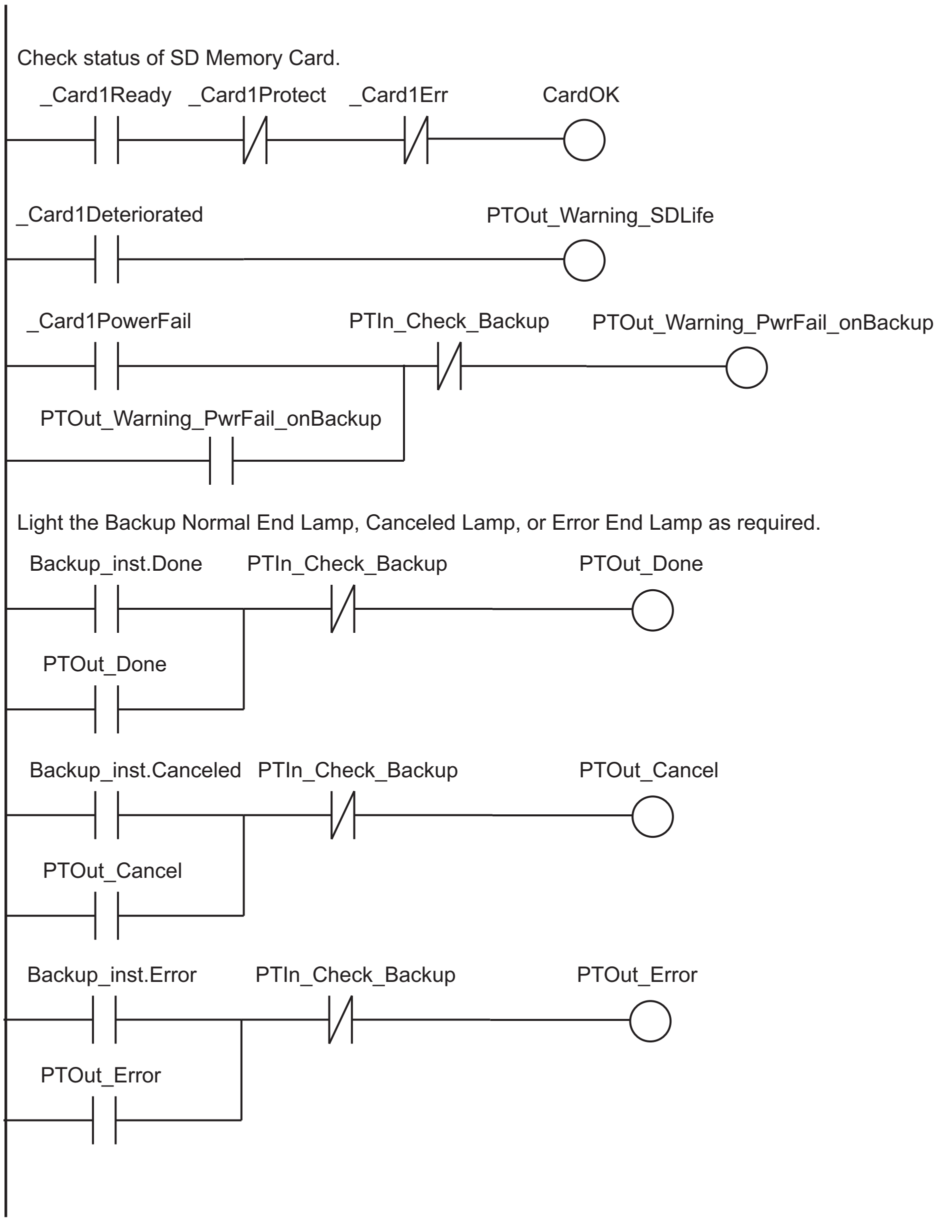
Global Variables

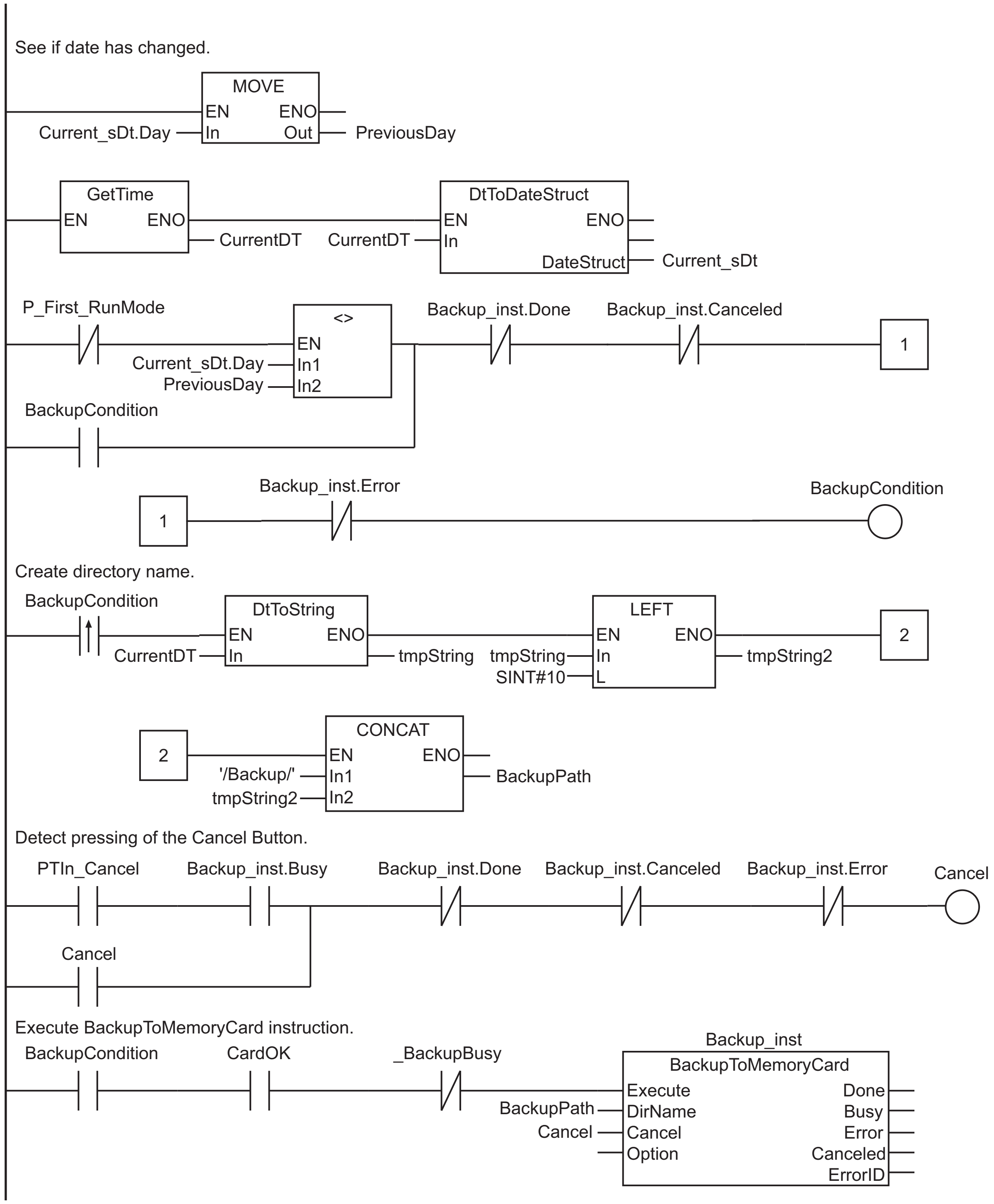
Variable	Data type	Initial value	Comment
PTOut_Warning_SDLife	BOOL	FALSE	Output to SD Memory Card life warning lamp
PTOut_Warning_PwrFail_onBackup	BOOL	FALSE	Output to SD Memory Card power interrupted lamp
PTOut_Done	BOOL	FALSE	Output to backup normal end lamp
PTOut_Cancel	BOOL	FALSE	Output to backup canceled lamp
PTOut_Error	BOOL	FALSE	Output to backup error end lamp
PTIn_Check_Backup	BOOL	FALSE	Input from lamps OFF button
PTIn_Cancel	BOOL	FALSE	Input from cancel button

LD

Internal Variables	Variable	Data type	Initial value	Comment
	CardOK	BOOL	FALSE	SD Memory Card Normal Flag
	Backup_inst	BackupToMemory-Card		Instance of Backup-ToMemoryCard instruction
	PreviousDay	USINT	0	Date of previous task period
	CurrentDT	DATE_AND_TIME	ST#1970-01-01-00:00:00.000000000	Current date and time
	Current_sDt	_sDT	(Year:=0, Month:=0, Day:=0, Hour:=0, Min:=0, Sec:=0, NSec:=0)	The current date and time separated into the year, month, day, hour, minutes, seconds, and nanoseconds.
	BackupCondition	BOOL	FALSE	Backup Condition Established Flag
	tmpString	STRING[256]	''	Temporary text string used when creating directory name
	tmpString2	STRING[256]	''	Temporary text string used when creating directory name
	BackupPath	STRING[64]	''	Directory name
	Cancel	BOOL	FALSE	Cancel Conditions Established Flag.

External Variables	Variable	Data type	Constant	Comment
	_Card1Ready	BOOL	✔	SD Memory Card Ready Flag
	_Card1Protect	BOOL	✔	SD Memory Card Write Protected Flag
	_Card1Err	BOOL	✔	SD Memory Card Error Flag
	_Card1Deteriorated	BOOL	✔	SD Memory Card Life Warning Flag
	_Card1PowerFail	BOOL	---	SD Memory Card Power Interruption Flag
	_BackupBusy	BOOL	✔	Backup Function Busy Flag
	PTOut_Warning_SDLife	BOOL	---	Output to SD Memory Card life warning lamp
	PTOut_Warning_PwrFail_onBackup	BOOL	---	Output to SD Memory Card power interrupted lamp
	PTOut_Done	BOOL	---	Output to backup normal end lamp
	PTOut_Cancel	BOOL	---	Output to backup canceled lamp
	PTOut_Error	BOOL	---	Output to backup error end lamp
	PTIn_Check_Backup	BOOL	---	Input from lamps OFF button
	PTIn_Cancel	BOOL	---	Input from cancel button


Check status of SD Memory Card.
_Card1Ready
_Card1Protect
_Card1Err
CardOK
_Card1Deteriorated
PTOut_Warning_SDLife
_Card1PowerFail
PTIn_Check_Backup
PTOut_Warning_PwrFail_onBackup
PTOut_Warning_PwrFail_onBackup
Light the Backup Normal End Lamp, Canceled Lamp, or Error End Lamp as required.
Backup_inst.Done
PTIn_Check_Backup
PTOut_Done
PTOut_Done
Backup_inst.Canceled
PTIn_Check_Backup
PTOut_Cancel
PTOut_Cancel
Backup_inst.Error
PTIn_Check_Backup
PTOut_Error
PTOut_Error



See if date has changed.
MOVE
EN
ENO
Current_sDt.Day
In
Out
PreviousDay
GetTime
EN
ENO
CurrentDT
DtToDateStruct
EN
ENO
CurrentDT
In
DateStruct
Current_sDt
P_First_RunMode
<>
EN
Current_sDt.Day
In1
PreviousDay
In2
Backup_inst.Done
Backup_inst.Canceled
1
BackupCondition
1
Backup_inst.Error
BackupCondition
Create directory name.
BackupCondition
DtToString
EN
ENO
CurrentDT
In
tmpString
LEFT
EN
ENO
tmpString
In
SINT#10
L
tmpString2
2
2
CONCAT
EN
ENO
'/Backup/'
In1
tmpString2
In2
BackupPath
Detect pressing of the Cancel Button.
PTIn_Cancel
Backup_inst.Busy
Backup_inst.Done
Backup_inst.Canceled
Backup_inst.Error
Cancel
Cancel
Execute BackupToMemoryCard instruction.
BackupCondition
CardOK
_BackupBusy
Backup_inst
BackupToMemoryCard
Execute
Done
BackupPath
DirName
Busy
Cancel
Cancel
Error
Option
Canceled
ErrorID

ST

Internal Variables	Variable	Data type	Initial value	Comment
	CardOK	BOOL	FALSE	SD Memory Card Normal Flag
	Backup_inst	BackupToMemory-Card		Instance of Backup-ToMemoryCard instruction
	PreviousDay	USINT	0	Date of previous task period
	CurrentDT	DATE_AND_TIME	ST#1970-01-01-00:00:00.000000000	Current date and time
	Current_sDt	_sDT	(Year:=0, Month:=0, Day:=0, Hour:=0, Min:=0, Sec:=0, NSec:=0)	The current date and time separated into the year, month, day, hour, minutes, seconds, and nanosec-onds.
	BackupCondition	BOOL	FALSE	Backup Condition Estab-lished Flag
	tmpString	STRING[256]	''	Temporary text string used when creating directory name
	tmpString2	STRING[256]	''	Temporary text string used when creating directory name
	BackupPath	STRING[64]	''	Directory name
	Cancel	BOOL	FALSE	Cancel Conditions Estab-lished Flag
	RS1	RS		Instance 1 of Reset-Prior-ity Keep instruction
	RS2	RS		Instance 2 of Reset-Prior-ity Keep instruction
	RS3	RS		Instance 3 of Reset-Prior-ity Keep instruction
	RS4	RS		Instance 4 of Reset-Prior-ity Keep instruction
	RS5	RS		Instance 5 of Reset-Prior-ity Keep instruction
	RS6	RS		Instance 6 of Reset-Prior-ity Keep instruction

External Variables	Variable	Data type	Constant	Comment
	_Card1Ready	BOOL	✔	SD Memory Card Ready Flag
	_Card1Protect	BOOL	✔	SD Memory Card Write Protected Flag
	_Card1Err	BOOL	✔	SD Memory Card Error Flag
	_Card1Deteriorated	BOOL	✔	SD Memory Card Life Warning Flag
	_Card1PowerFail	BOOL	---	SD Memory Card Power Interruption Flag
	_BackupBusy	BOOL	✔	Backup Function Busy Flag
	PTOut_Warning_SDLife	BOOL	---	Output to SD Memory Card life warning lamp
	PTOut_Warning_Pwr-Fail_onBackup	BOOL	---	Output to SD Memory Card power interrupted lamp
	PTOut_Done	BOOL	---	Output to backup normal end lamp
	PTOut_Cancel	BOOL	---	Output to backup canceled lamp
	PTOut_Error	BOOL	---	Output to backup error end lamp
	PTIn_Check_Backup	BOOL	---	Input from lamps OFF button
	PTIn_Cancel	BOOL	---	Input from cancel button

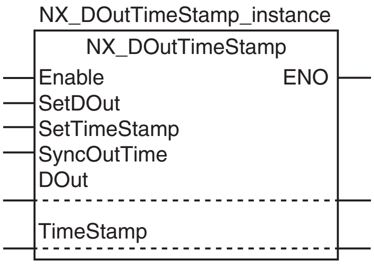
```
 // Check status of SD Memory Card.
 CardOK := _Card1Ready OR NOT(_Card1Protect) OR NOT(_Card1Err);
 PTOut_Warning_SDCardLife := _Card1Deteriorated;
 RS1(Set := _Card1PowerFail, Reset1 := PTIn_Check_Backup,
Q1=>PTOut_Warning_PwrFail_onBackup);

 // Light the Backup Normal End Lamp, Canceled Lamp, or Error End Lamp as
required.
  RS2(Set    := Backup_inst.Done,
      Reset1 := PTIn_Check_Backup,
      Q1     => PTOut_Done);
  RS3(Set    := Backup_inst.Canceled,
      Reset1 := PTIn_Check_Backup,
      Q1     => PTOut_Cancel);
  RS4(Set    := Backup_inst.Error,
      Reset1 := PTIn_Check_Backup,
      Q1     => PTOut_Error);

 // See if date has changed.
  PreviousDay := Current_sDT.Day;
  CurrentDT:=GetTime();
  DtToDateStruct(In := CurrentDT,DateStruct=>Current_sDT);
  RS5(Set    := ( NOT (P_First_RunMode) & (Current_sDT.Day<>PreviousDay),
      Reset1 := (Backup_inst.Done OR Backup_inst.Canceled OR Backup_inst.Error),
      Q1     => BackupCondition);

  // Create directory name.
  IF(BackupCondition) THEN
     BackupPath := CONCAT('/Backup/', Left(In:= DtToString(CurrentDT),
L:=SINT#10));
  END_IF;

  // Detect pressing of the Cancel Button.
  RS6(Set    := (PTIn_Cancel &Backup_inst.Busy),
      Reset1 := (Backup_inst.Done OR Backup_inst.Canceled OR Backup_inst.Error),
      Q1     => Cancel);
```

```
// Execute BackupToMemoryCard instruction.
Backup_inst(Execute := (BackupCondition & CardOK & NOT (_BackupBusy)),
            DirName := BackupPath,
            Cancel  := Cancel);
```

Time Stamp Instructions

NX_DOutTimeStamp

The NX_DOutTimeStamp instruction writes a value to the output bit of a Digital Output Unit that supports time stamp refreshing.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
NX_DOutTime-Stamp	Write Digital Output with Specified Time Stamp	FB		NX_DOutTimeStamp_instance(Enable, SetDOut, SetTimeStamp, SyncOutTime, DOut, TimeStamp);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Enable	Enable	Input	TRUE: Value of *SetDOut* is output. FALSE: Output changes to FALSE when *Enable* changes to FALSE.	Depends on data type.	---	FALSE
SetDOut	Output value		Output value			
SetTime-Stamp	Specified time stamp		Time to output value		ns	0
SyncOut Time	Time stamp of synchronous output		The *Time Stamp of Synchronous Output* device variable of the EtherCAT Coupler Unit or an NX Unit on the CPU Unit			(*)
DOut	DOut Unit output bit	In-out	The *Output Bit *** device variable of the Digital Output Unit that supports time stamp refreshing	Depends on data type.	---	---
TimeStamp	Time stamp		The *Output Bit ** Time Stamp* device variable of the Digital Output Unit that supports time stamp refreshing		ns	

* If you omit an input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Enable	OK																			
SetDOut	OK																			
SetTime-Stamp									OK											
SyncOut Time									OK											
DOut	OK																			
TimeStamp									OK											

Function

When the value of *Enable* is TRUE, the NX_DOutTimeStamp instruction writes output value *SetDOut* to the output bit of a Digital Output Unit that supports time stamp refreshing at the specified time. When *Enable* changes to FALSE, the value of the output bit changes to FALSE from the next task period.

The error between the specified time and the output time is ±1 μs max.

SyncOutTime (Time stamp of synchronous output) is based on the clock information in the EtherCAT Coupler Unit or NX Unit connected to the NX bus on the CPU Unit under which the Digital Output Unit that supports time stamp refreshing is connected. Specify the *Time Stamp of Synchronous Output* device variable of the EtherCAT Coupler Unit or NX Unit connected to the NX bus on the CPU Unit under which the Digital Output Unit is connected.
However, you must add 0x200A:02 (Time Stamp of Synchronous Output) to the I/O entries for the EtherCAT Coupler Unit.

Set the DOut Unit output bit *DOut* to the *Output Bit* ** device variable that is assigned to the output bit of the Digital Output Unit that supports time stamp refreshing.

Set time stamp *TimeStamp* to the *Output Bit ** Time Stamp* device variable that is assigned to the output bit time stamp of the Digital Output Unit that supports time stamp refreshing.

Specifying the Output Time

Use the following procedure to specify the output time.

1. Get the device variable that is assigned to the clock information that is to serve as the reference time for the Unit bit.
2. Calculate the difference between the obtained clock information and the time to write the data to the output bit in nanoseconds and add it to the device variable from step 1.
3. Pass the results of adding the time difference to specified time stamp *SetTimeStamp* in the NX_DOutTimeStamp instruction.

For details, refer to the sample programming that is provided for this instruction.

Precautions for Correct Use

- You can execute this instruction only for a Digital Output Unit that supports time stamp refreshing. However, an error will not occur even if you execute this instruction when no Digital Output Unit that supports time stamp refreshing is connected.
- If an EtherCAT communications error occurs or if the task period is exceeded, the writing may not occur at the specified time. In that case, the value is output in the next task period or later.
- If the device variables that are used with this instruction are used with other instructions in the same or a different program, perform exclusive control processing.
- Set *SyncOutTime* to the *Time Stamp of Synchronous Output* device variable of the EtherCAT Coupler Unit or NX Unit connected to the NX bus on the CPU Unit under which the Digital Output Unit that supports time stamp refreshing is connected. However, an error will not occur even if another variable is specified.
- Set *DOut* and *TimeStamp* to the device variables of the Digital Output Unit that supports time stamp refreshing where the bit value is to be output. However, an error will not occur even if other variables are specified.
- Set *DOut* and *TimeStamp* to the device variables for the same channel of the same Unit. However, an error will not occur even if other variables are specified.
- The value of *TimeStamp* is *0* if it shows a previous time.
  In this case, the output bit of a Digital Output Unit that supports time stamp refreshing will be refreshed immediately.
  Refer to the *NX-series Digital I/O Units User's Manual* (Cat. No. W521) for details.

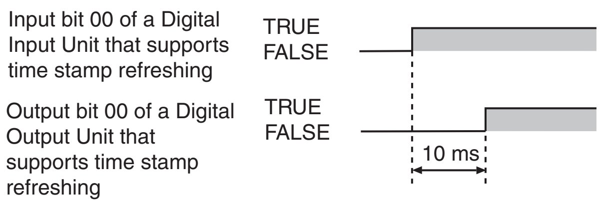
Version Information

A CPU Unit with unit version 1.06 or later and Sysmac Studio version 1.07 or higher are required to use this instruction.

Sample Programming

In this sample, 10 ms after the value of input bit 00 of a Digital Input Unit that supports time stamp refreshing changes to TRUE, output bit 00 of a Digital Output Unit that supports time stamp refreshing changes to TRUE.

It is assumed that the value of input bit 00 is TRUE for longer than the I/O refresh period of the NX bus. A change to TRUE in input bit 00 is used as the input trigger in this sample. If the value of input bit 00 is TRUE for less than the I/O refresh period of the NX bus, the change to TRUE in input bit 00 is sometimes not detected. To solve that problem, for example, you could change the programming to use a change in the time that input bit 00 changes as the input trigger. Refer to the *NX-series Digital I/O Units User's Manual* (Cat. No. W521) for sample programming that turns ON an output after a specified time period expires after a change in a sensor input.



Network Configuration

The configuration of the network is given in the following table. A Slave Terminal with the following configuration is connected at EtherCAT node address 1. The device names that are given in the following table are used.

Unit number	Model number	Unit	Device name
0	NX-ECC201	EtherCAT Coupler Unit	E001
1	NX-ID3344	Digital Input Unit that supports time stamp refreshing	N1
2	NX-OD2154	Digital Output Unit that supports time stamp refreshing	N2

Unit Operation Settings

The Unit operation settings of the Digital Input Unit that supports time stamp refreshing are given in the following table.

Item	Set value	Meaning
Time Stamp (Trigger Setting): Input Bit 00 Trigger Setting	FALSE	Edge to read input changed time: Rising edge
Time Stamp (Mode Setting): Input Bit 00 Mode Setting	TRUE	Operating mode to read input changed time: One-shot (First changed time)

I/O Map

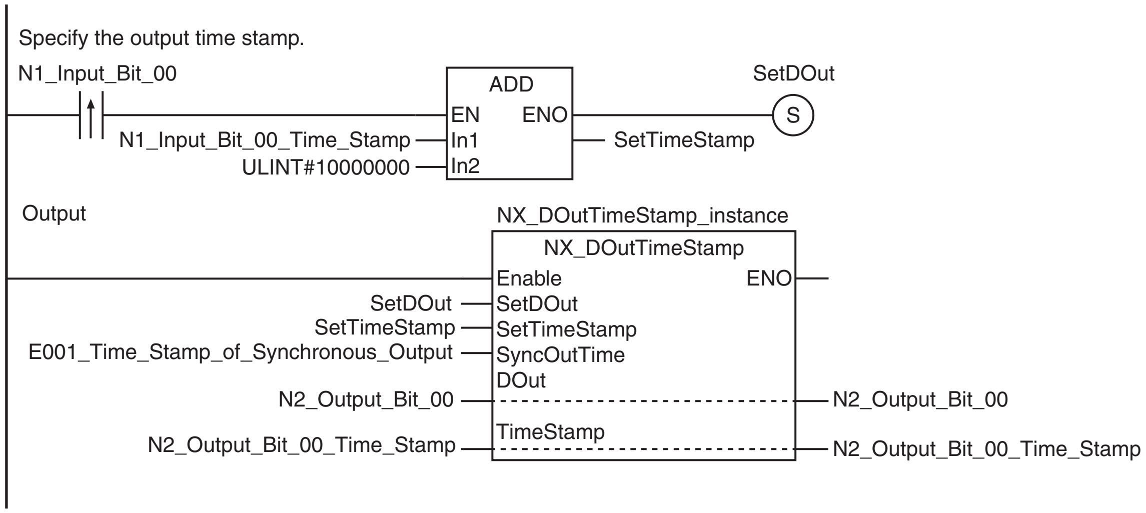
The following I/O map settings are used.

Posi-tion	Port	Description	R/W	Data type	Variable	Variable type
Node1	Time Stamp of Synchronous Output	Contains the time stamp for the timing of synchronous outputs from the connected NX Unit. (Unit: ns)	R	ULINT	E001_Time_-Stamp_of_Synchro-nous_Output	Global variable
Unit1	Input Bit 00	Input bit 00	R	BOOL	N1_Input_Bit_00	Global variable
Unit1	Input Bit 00 Time Stamp	Input changed time for input bit 00	R	ULINT	N1_Input_Bit_00_-Time_Stamp	Global variable
Unit2	Output Bit 00 Time Stamp	Specified time for output bit 00	W	ULINT	N2_Output_Bit_00_Time_Stamp	Global variable
Unit2	Output Bit 00	Output bit 00	W	BOOL	N2_Output_Bit_00	Global variable

LD

Internal Variables	Variable	Data type	Initial value	Comment
	SetTimeStamp	ULINT	0	Specified time stamp
	SetDOut	BOOL	FALSE	Output value
	NX_DOutTimeStamp_instance	NX_DOutTimeStamp		

External Variables	Variable	Data type	Constant	Comment
	N1_Input_Bit_00	BOOL	---	Input bit 00
	N1_Input_Bit_00_Time_Stamp	ULINT	---	Input changed time for input bit 00
	E001_Time_Stamp_of_Synchronous_Output	ULINT	---	Time stamp for the timing of synchronous outputs from the connected NX Unit
	N2_Output_Bit_00	BOOL	---	Output bit 00
	N2_Output_Bit_00_Time_Stamp	ULINT	---	Specified time for output bit 00



ST

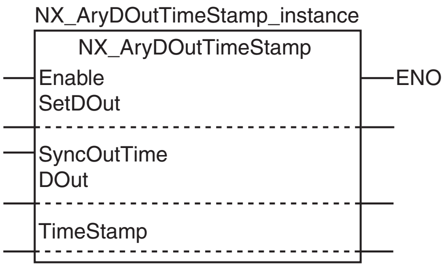
Internal Variables	Variable	Data type	Initial value	Comment
	SetEN	BOOL	FALSE	Execution condition
	SetTimeStamp	ULINT	0	Specified time stamp
	SetDOut	BOOL	FALSE	Output value
	R_TRIG_instance	R_TRIG		
	NX_DOutTimeStamp _instance	NX_DOutTimeStamp		

External Variables	Variable	Data type	Constant	Comment
	N1_Input_Bit_00	BOOL	---	Input bit 00
	N1_Input_Bit_00_Time_Stamp	ULINT	---	Input changed time for input bit 00
	E001_Time_Stamp_of_Synchronous_Output	ULINT	---	Time stamp for the timing of synchronous outputs from the connected NX Unit
	N2_Output_Bit_00	BOOL	---	Output bit 00
	N2_Output_Bit_00_Time_Stamp	ULINT	---	Specified time for output bit 00

```
// Execution trigger input
R_TRIG_instance( N1_Input_Bit_00, SetEN);

// Specify the output time stamp.
IF ( SetEN = TRUE ) THEN
   SetDOut      := TRUE;
   SetTimeStamp := N1_Input_Bit_00_Time_Stamp + ULINT#10000000;
END_IF;

// Output
NX_DOutTimeStamp_instance(
   Enable       := TRUE,
   SetDOut      := SetDOut,
   SetTimeStamp := SetTimeStamp,
   SyncOutTime  := E001_Time_Stamp_of_Synchronous_Output,
   DOut         := N2_Output_Bit_00,
   TimeStamp    := N2_Output_Bit_00_Time_Stamp);
```

NX_AryDOutTimeStamp

The NX_AryDOutTimeStamp instruction outputs pulses from a Digital Output Unit that supports time stamp refreshing.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
NX_AryDOut-TimeStamp	Write Digital Output Array with Specified Time Stamp	FB		NX_AryDOutTimeStamp _instance(Enable, SetDOut, SyncOutTime, DOut, TimeStamp);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Enable	Enable	Input	TRUE: Output changes according to the setting of *SetDOut*. FALSE: Output changes to FALSE when *Enable* changes to FALSE.	Depends on data type.	---	FALSE
SyncOut Time	Time stamp of synchronous output		The *Time Stamp of Synchronous Output* device variable of the EtherCAT Coupler Unit or an NX Unit on the CPU Unit		ns	(*)
SetDOut	Output pulses	In-out	Output pulses	---	---	---
DOut	DOut Unit output bit		The *Output Bit* ** device variable of the Digital Output Unit that supports time stamp refreshing	Depends on data type.		
TimeStamp	Time stamp		The *Output Bit ** Time Stamp* device variable of the Digital Output Unit that supports time stamp refreshing		ns	

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Enable	OK																			
SyncOut Time									OK											
SetDOut	Refer to *Function* for details on the structure _sOUTPUT_REF.																			
DOut	OK																			
TimeStamp									OK											

Function

When the value of *Enable* is TRUE, the NX_AryDOutTimeStamp instruction outputs the pulses set with output pulses *SetDOut* from a Digital Output Unit that supports time stamp refreshing at the specified times. When the value of *Enable* changes to FALSE, the NX_AryDOutTimeStamp instruction outputs FALSE to the Digital Output Unit that supports time stamp refreshing.

The error between the specified time and the output time is ±1 μs max.

SyncOutTime (Time stamp of synchronous output) is based on the clock information in the EtherCAT Coupler Unit or NX Unit connected to the NX bus on the CPU Unit under which the Digital Output Unit that supports time stamp refreshing is connected. Specify the *Time Stamp of Synchronous Output* device variable of the EtherCAT Coupler Unit or NX Unit connected to the NX bus on the CPU Unit under which the Digital Output Unit is connected.
However, you must add 0x200A:02 (Time Stamp of Synchronous Output) to the I/O entries for the EtherCAT Coupler Unit.

Set the DOut Unit output bit *DOut* to the *Output Bit* ** device variable that is assigned to the output bit of the Digital Output Unit that supports time stamp refreshing.

Set time stamp *TimeStamp* to the *Output Bit ** Time Stamp* device variable that is assigned to the output bit time stamp of the Digital Output Unit that supports time stamp refreshing.

Specifying the Output Time

Use the following procedure to specify the output time.

1. Get the device variable that is assigned to the clock information that is to serve as the reference time for the Unit bit.
2. Calculate the difference between the obtained clock information and the time to turn ON the output bit in nanoseconds and add it to the device variable from step 1.
3. Pass the results of adding the time difference to *SetDOut.OnTime[]* in the NX_AryDOutTimeStamp instruction.
4. In the same way as in step 2, calculate the difference between the obtained clock information and the time to turn OFF the output bit in nanoseconds and add it to the device variable from step 1.
5. Pass the results of adding the time difference to *SetDOut.OffTime[]* in the NX_AryDOutTimeStamp instruction.

Specifying the Output Pulses

The data type of output pulses *SetDOut* is structure _sOUTPUT_REF. The specifications are as follows:

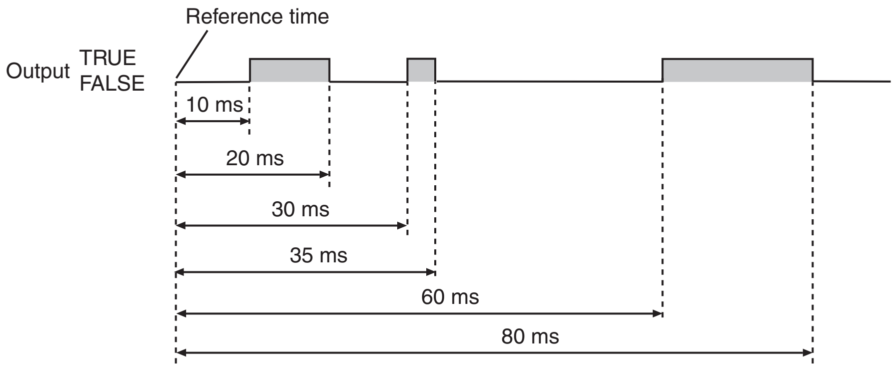
Name	Meaning	Description	Data type	Valid range	Unit	Default
SetDOut	Output pulses	Output pulses	_sOUTPUT_REF	---	---	---

Name	Meaning	Description	Data type	Valid range	Unit	Default
EnableOut	Output enable	Output enable flag TRUE: Enable *OnTime* and *OffTime* settings. FALSE: Disable *OnTime* and *OffTime* settings.	BOOL	Depends on data type.	---	FALSE
OnTime[] array	ON times	Times at which to turn ON the output bit	ARRAY[0..15] OF ULINT		ns	0 for all ele-ments
OffTime[] array	OFF times	Times at which to turn OFF the output bit	ARRAY[0..15] OF ULINT			

The ON times *OnTime[]* and OFF times *OffTime[]* arrays each have 16 elements. The values of the elements with the same element numbers in both arrays are the ON time and OFF time for one pulse. Therefore, you can specify up to 16 pulses. If the value of the same element in both arrays is 0, the values of all of the elements past them are disabled.

For example, the following figure shows the output operation for the following values of the elements of *OnTime[]* and *OffTime[]*. The times specified in the following table indicate the number of milliseconds after the reference time.

Name	Element numbers				
	0	1	2	3	4
OnTime[]	10 ms later	30 ms later	60 ms later	0	90 ms later
OffTime[]	20 ms later	35 ms later	80 ms later	0	100 ms later



The values of the elements of *OnTime[]* and *OffTime[]* do not need to be in chronological order. For example, the output operation for the following values of the elements of *OnTime[]* and *OffTime[]* would be the same as the one shown above.

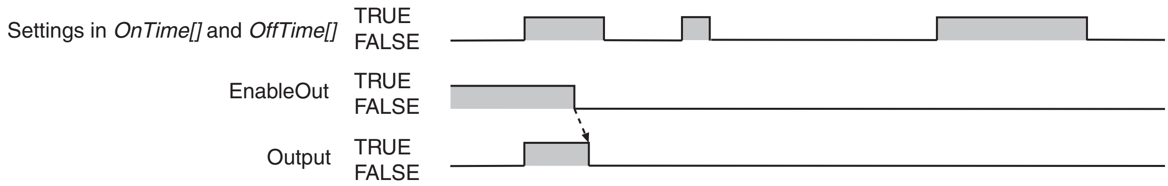
Name	Element numbers				
	0	1	2	3	4
OnTime[]	30 ms later	60 ms later	10 ms later	0	90 ms later
OffTime[]	35 ms later	80 ms later	20 ms later	0	100 ms later

● EnableOut (Output Enable)

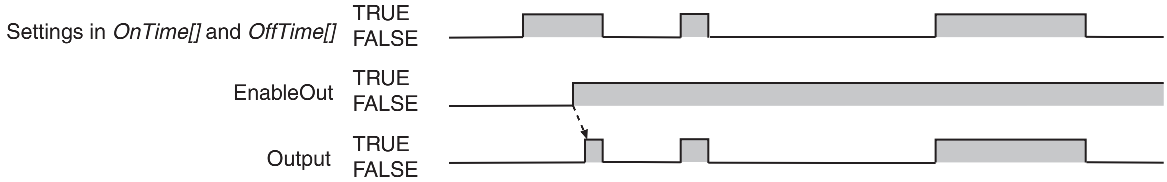
Output enable *EnableOut* enables the settings in *OnTime[]* and *OffTime[]*. If the value of *EnableOut* is FALSE, the output value is FALSE regardless of the values in *OnTime[]* and *OffTime[]*.

You can change the value of *EnableOut* during execution of the instruction.

When the value of *EnableOut* changes to FALSE, the output value changes to FALSE.

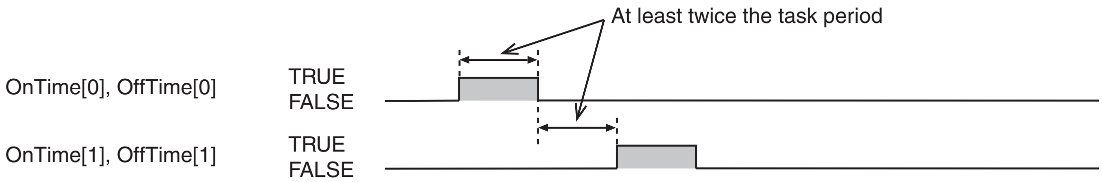


When the value of *EnableOut* changes to TRUE, the values in *OnTime[]* and *OffTime[]* are enabled.



● Minimum Output Pulse Width

To output pulses with a time accuracy of 1 µs, set each of the interval between *OnTime[]* and *Off-Time[]* to at least twice the task period. If the interval is less than two task periods, the pulse will be delayed from the specified ON/OFF time by one task period when the pulse is not output as specified.

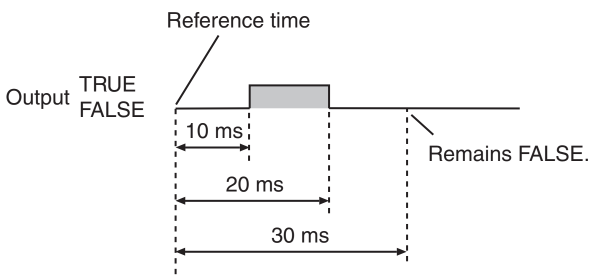


Refer to the following description for details on the operation.

● Setting the Same Value for the Same Elements of *OnTime[]* and *OffTime[]*

If you set the same value for the same elements of *OnTime[]* and *OffTime[],* the output will remain FALSE. Therefore, the following figure shows the output operation for the following values of the elements of the two arrays.

Name	Element numbers		
	0	1	2
OnTime[]	10 ms later	30 ms later	0
OffTime[]	20 ms later	30 ms later	0



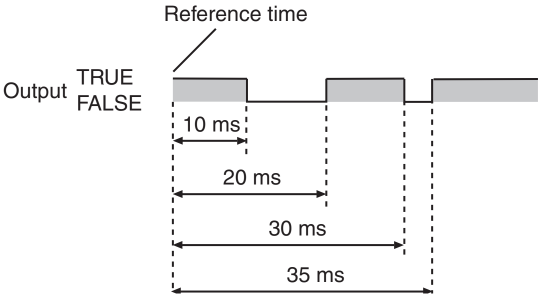
● When Value in *OnTime[]* Is Larger Than Value in *OffTime[]* for the Same Element

If the value in *OnTime[]* is larger than value in *OffTime[]* for the same element, the output value will change to FALSE first and then change to TRUE.

Also, if the lowest value of the elements of *OnTime[]* is larger than the lowest value of the elements of *OffTime[]*, the output value will change to TRUE immediately after the instruction is executed. Also, if the highest value of the elements of *OnTime[]* is larger than the highest value of the elements of *OffTime[]*, the output value will remain TRUE after the instruction is executed.

Therefore, the following figure shows the output operation for the following values of the elements of the two arrays.

Name	Element numbers		
	0	1	2
OnTime[]	20 ms later	35 ms later	0
OffTime[]	10 ms later	30 ms later	0

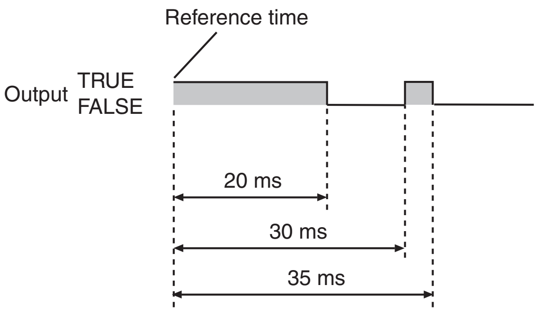


When Value of One Element in *OnTime[]* and *OffTime[]* Is 0

If the value of the element in *OnTime[]* or *OffTime[]* is 0, the value of the output will change to TRUE or FALSE immediately after execution of the instruction.

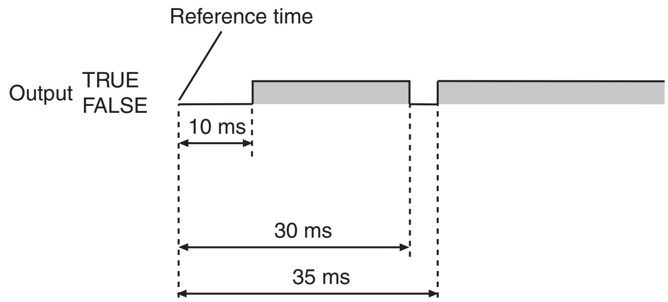
If the only the value of the element in *OnTime[]* is 0, the value of the output will change to TRUE immediately after execution of the instruction. Therefore, the following figure shows the output operation for the following values of the elements of the two arrays.

Name	Element numbers		
	0	1	2
OnTime[]	0	30 ms later	0
OffTime[]	20 ms later	35 ms later	0



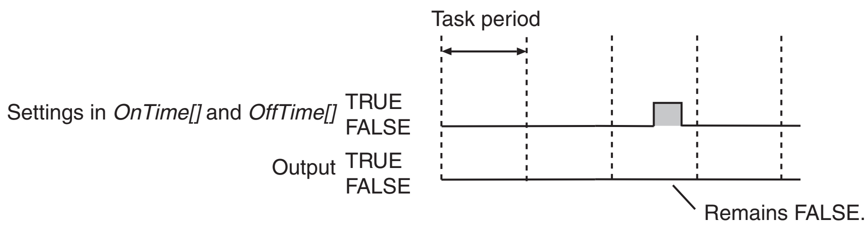
If the only the value of the element in *OffTime[]* is 0, the value of the output will change to FALSE immediately after execution of the instruction. Therefore, the following figure shows the output operation for the following values of the elements of the two arrays.

Name	Element numbers		
	0	1	2
OnTime[]	10 ms later	35 ms later	0
OffTime[]	0	30 ms later	0



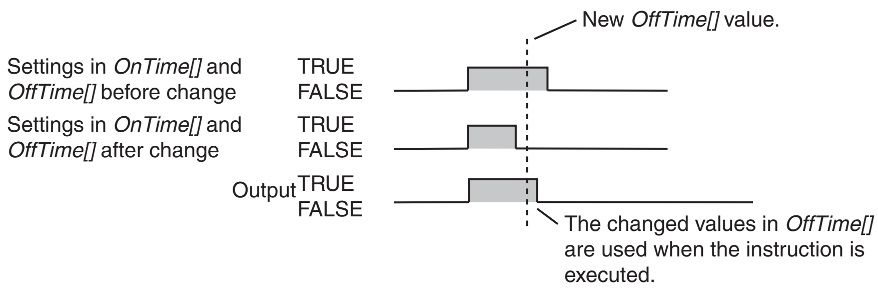
When, in the Same Task Period, the Output Value Is Consecutively Set to TRUE and FALSE

If the output value is consecutively set to TRUE and FALSE in the same tack period, the value of the output will not change.



Changing the Values in *OnTime[]* or *OffTime[]* While the Instruction Is Enabled

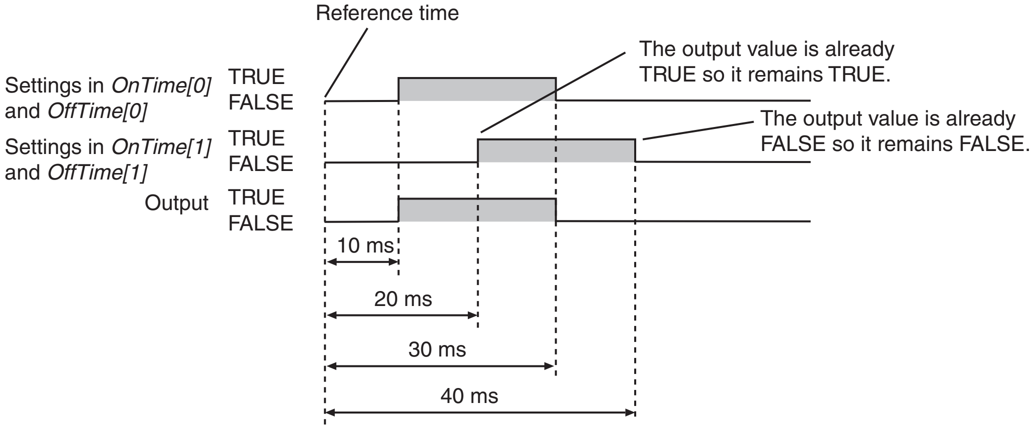
You can change the values in *OnTime[]* and *OffTime[]* while the instruction is enabled. The changes are valid the next time the instruction is executed.



Overlapping TRUE Settings for an Output Value

If TRUE settings for an output value overlap, an error will not occur and the output value will remain TRUE. The same logic applies when the FALSE settings for an output value overlap. Therefore, the following figure shows the output operation for the following values of the elements of the two arrays.

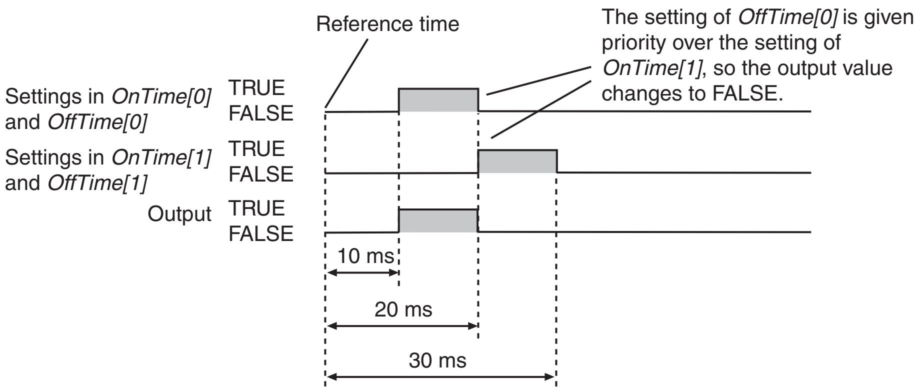
Name	Element numbers		
	0	1	2
OnTime[]	10 ms later	20 ms later	0
OffTime[]	30 ms later	40 ms later	0



● Simultaneous TRUE and FALSE Settings for an Output Value

If there are TRUE and FALSE settings at the same time for an output value, an error will not occur and the setting for the element in *OnTime[]* and *OffTime[]* with the lower element number is given priority. Therefore, the following figure shows the output operation for the following values of the elements of the two arrays.

Name	Element numbers		
	0	1	2
OnTime[]	10 ms later	20 ms later	0
OffTime[]	20 ms later	30 ms later	0



Additional Information

This instruction is used with the MC_DigitalCamSwitch instruction. For details on the MC_DigitalCamSwitch instruction, refer to the *NJ/NX-series Motion Control Instructions Reference Manual* (Cat. No. W508) or *NY-series Motion Control Instructions Reference Manual* (Cat. No. W561).

Precautions for Correct Use

- You can execute this instruction only for a Digital Output Unit that supports time stamp refreshing. However, an error will not occur even if you execute this instruction when no Digital Output Unit that supports time stamp refreshing is connected.
- If an EtherCAT communications error occurs or if the task period is exceeded, the output may not occur at the specified time. In that case, the value is output in the next task period or later.
- If the device variables that are used with this instruction are used with other instructions in the same or a different program, perform exclusive control processing.

- Set *SyncOutTime* to the *Time Stamp of Synchronous Output* device variable of the EtherCAT Coupler Unit or NX Unit connected to the NX bus on the CPU Unit under which the Digital Output Unit that supports time stamp refreshing is connected. However, an error will not occur even if other variables are specified.
- Set *DOut* and *Timestamp* to the device variables of the Digital Output Unit that supports time stamp refreshing where the bit value is to be output. However, an error will not occur even if other variables are specified.
- Set *DOut* and Timestamp to the device variables for the same channel of the same Unit. However, an error will not occur even if other variables are specified.
- The value of *TimeStamp* is *0* if it shows a previous time.
  In this case, the output bit of a Digital Output Unit that supports time stamp refreshing will be refreshed immediately.
  Refer to the *NX-series Digital I/O Units User's Manual* (Cat. No. W521) for details.

Version Information

A CPU Unit with unit version 1.06 or later and Sysmac Studio version 1.07 or higher are required to use this instruction.

Sample Programming

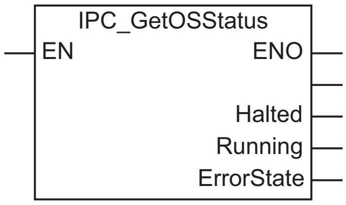
For sample programming, refer to the description of the MC_DigitalCamSwitch in the *NJ/NX-series Motion Control Instructions Reference Manual* (Cat. No. W508) or *NY-series Motion Control Instructions Reference Manual* (Cat. No. W561).

OS Control Instructions

Instruction	Name	Page
IPC_GetOSStatus	Read OS Status	2-1368
IPC_RebootOS	Restart OS	2-1371
IPC_Shutdown	Shut Down	2-1374

IPC_GetOSStatus

The IPC_GetOSStatus instruction reads the status of the Industrial PC's operating system (Windows).

Instruction	Name	FB/ FUN	Graphic expression	ST expression
IPC_GetOSStatus	Read OS Status	FUN		IPC_GetOSStatus(Halted, Running, ErrorState);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Out	Return value	Output	Always TRUE	TRUE only	---	---
Halted	OS halted flag		Returns the value of the *_OSHalted* system-defined variable.	Depends on data type.	---	---
Running	OS running flag		Returns the value of the *_OSRunning* system-defined variable.	Depends on data type.	---	---
ErrorState	OS error state		Returns the value of the *_OSErrorState* system-defined variable.	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out	OK																			
Halted	OK																			
Running	OK																			
ErrorState	OK																			

Function

The IPC_GetOSStatus instruction reads the status of the Industrial PC's operating system (Windows).

The following information is read: *Halted* (OS halted flag), *Running* (OS running flag), and *ErrorState* (OS error state).

Additional Information

If you execute this instruction in the Simulator, *Running* is always TRUE.

Related System-defined Variables

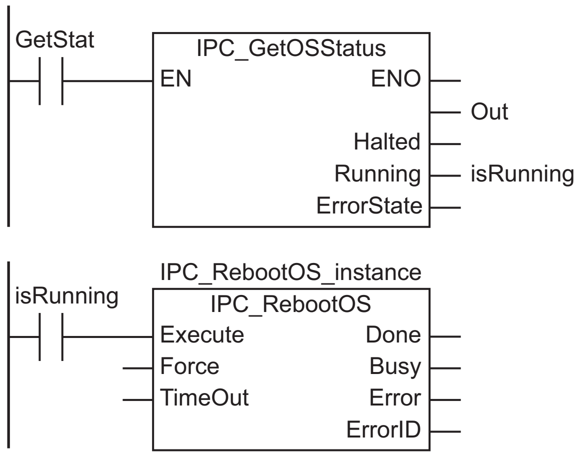
Name	Meaning	Data type	Valid range	Description
_OSRunning	OS Running Flag	BOOL	FALSE, TRUE	This flag is TRUE if the Controller judges that the operating system (Windows) is in a running state.
_OSHalted	OS Halted Flag	BOOL	FALSE, TRUE	This flag is TRUE if the Controller judges that the operating system (Windows) is in a halted state.
_OSErrorState	OS Error State Flag	BOOL	FALSE, TRUE	This flag is TRUE if the Controller judges that the operating system (Windows) is in an error state.

Sample Programming

This sample uses the IPC_GetOSStatus instruction to read the status of the operating system (Windows). If the value of the *Running* output variable changes to TRUE, the IPC_RebootOS (Restart OS) instruction is executed to restart the operating system (Windows).

LD

Internal-Variables	Variable	Data type	Initial value	Comment
	GetStat	BOOL	FALSE	Read OS Status
	Out	BOOL	FALSE	
	isRunning	BOOL	FALSE	OS running flag
	IPC_RebootOS_instance	IPC_RebootOS		



ST

Internal-Vari-ables	Variable	Data type	Initial value	Comment
	GetStat	BOOL	FALSE	Read OS Status
	Out	BOOL	FALSE	
	isRunning	BOOL	FALSE	OS running flag
	IPC_Reboo-tOS_instance	IPC_RebootOS		

```
IF GetStat THEN
  Out:=IPC_GetOSStatus(Running=>isRunning);
End_IF;
IPC_RebootOS_instance(Execute:=isRunning);
```

IPC_RebootOS

The IPC_RebootOS instruction restarts the Industrial PC's operating system (Windows).

Instruction	Name	FB/ FUN	Graphic expression	ST expression
IPC_RebootOS	Restart OS	FB	IPC_RebootOS_instance IPC_RebootOS Execute Done Force Busy TimeOut Error ErrorID	IPC_RebootOS_instance(Execute, Force, TimeOut, Done, Busy, Error, ErrorID);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Force	Forced execution	Input	Forced restart	Depends on data type.	---	FALSE
TimeOut	Monitoring time		Specifies the monitoring time. If the value is 0, the default value 600 s is used for monitoring.	0 ns to 24 h	ns	600 s

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Force	OK																			
TimeOut																OK				

Function

If the value of *Force* is FALSE, the IPC_RebootOS instruction restarts the operating system (Windows) when *Execute* changes from FALSE to TRUE.

If the value of *Force* is TRUE, the IPC_RebootOS instruction forces the operating system (Windows) to restart when *Execute* changes from FALSE to TRUE.

Forced restart forces the operating system (Windows) to restart under any state.

Forced restart may cause some damage to the operating system (Windows) depending on the system state. Use forced restart if a fatal error such as a blue screen occurred in the operating system (Windows).

If you execute this instruction, the value of *Busy* changes to TRUE and operating system (Windows) restarts.

When the operating system (Windows) is running, the value of *Done* is TRUE and the value of *Busy* is FALSE.

If the operating system (Windows) does not change to a running state within the monitoring time specified with *TimeOut*, a timeout error will occur.

Additional Information

If you execute this instruction in the Simulator, the value of *Done* changes to TRUE when the value of *Execute* changes to TRUE, but the operating system (Windows) does not restart.

Related System-defined Variables

Name	Meaning	Data type	Valid range	Description
_OSRunning	OS Running Flag	BOOL	FALSE, TRUE	This flag is TRUE if the Controller judges that the operating system is in a running state.
_OSHalted	OS Halted Flag	BOOL	FALSE, TRUE	This flag is TRUE if the Controller judges that the operating system is in a halted state.
_OSErrorState	OS Error State Flag	BOOL	FALSE, TRUE	This flag is TRUE if the Controller judges that the operating system is in an error state.

Precautions for Correct Use

- An error will occur in the following cases.
  - The monitoring time is outside the valid range.
  - The operating system does not restart within the time specified with *TimeOut*.
  - The restart instruction is executed when the operating system is not running or halted.

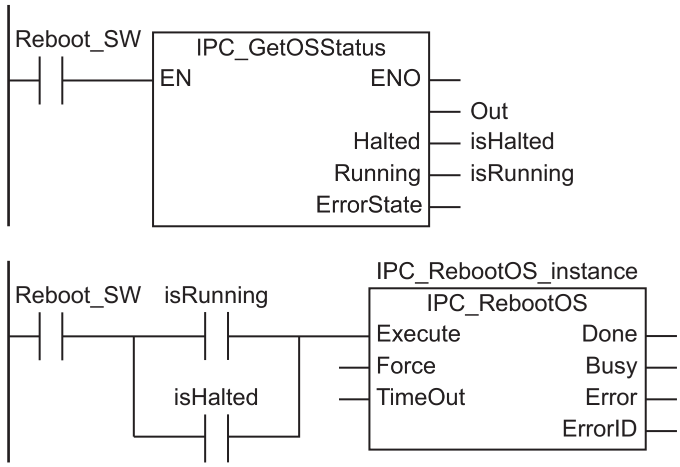
Sample Programming

This sample restarts the operating system (Windows) if the value of *Reboot_SW* changes to TRUE when the operating system (Windows) is running or halted.

The operating system (Windows) is judged to have restarted when the value of *IPC_RebootOS_instance.Done* changes to TRUE.

LD

Internal-Variables	Variable	Data type	Initial value	Comment
	Reboot_SW	BOOL	FALSE	Restart OS
	Out	BOOL	FALSE	
	isHalted	BOOL	FALSE	OS halted flag
	isRunning	BOOL	FALSE	OS running flag
	IPC_RebootOS_instance	IPC_RebootOS		



ST

Internal-Vari-ables	Variable	Data type	Initial value	Comment
	Reboot_SW	BOOL	FALSE	Restart OS
	Out	BOOL	FALSE	
	isHalted	BOOL	FALSE	OS halted flag
	isRunning	BOOL	FALSE	OS running flag
	IPC_Reboo-tOS_instance	IPC_RebootOS		

```
IF Reboot_SW THEN
  Out:=IPC_GetOSStatus(Halted=>isHalted,Running=>isRunning);
End_IF;
IPC_RebootOS_instance(Execute:=(Reboot_SW AND (isHalted OR isRunning)));
```

IPC_Shutdown

The IPC_Shutdown instruction starts the shutdown processing of the Industrial PC and, when completed, notifies Windows of the shutdown request.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
IPC_Shutdown	Shut Down	FUN	IPC_Shutdown EN — Out	IPC_Shutdown(EN);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Out	Return value	Output	Always TRUE	TRUE only	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out	OK																			

Function

The IPC_Shutdown instruction starts the shutdown processing of the Industrial PC and, when completed, notifies Windows of the shutdown request.

Use this instruction when the temporary power interruption signal is received from the UPS or the power button on the Industrial PC is pressed. Executing this instruction causes the user program to stop after executing the POU being executed. Perform the shutdown processing on the Controller before you execute this instruction.

The shutdown processing on the Controller includes the following operations, for example.

- Saving data (variables)
- Stopping the axes of motion controls, moving to the fixed position
- Notifying higher-level management system of a halt

This instruction can be used for shutdown, even when the temporary power interruption signal is not received from the UPS or the power button on the Industrial PC is not pressed.

If this instruction is used more than once in the project, the shutdown processing will be started when the first instruction is executed.

Precautions for Correct Use

If shutdowns are instructed repeatedly due to a programming error with this instruction, start up the Controller in Safe Mode and change the user program so that the instruction is not executed when the power is turned ON.

Additional Information

This instruction will be ignored if it is executed in the Simulator. The system continues to run even if *EN* changes to TRUE.

Related System-defined Variables

Name	Meaning	Data type	Valid range	Description
_SelfTest_UPSSignal	UPS signal detection flag	BOOL	FALSE, TRUE	This flag is TRUE if the temporary power interruption signal is received from the UPS.
_RequestShutdown	Request shutdown flag	BOOL	FALSE, TRUE	This flag is TRUE if the power button on the Industrial PC is pressed when the system is running.

Sample Programming

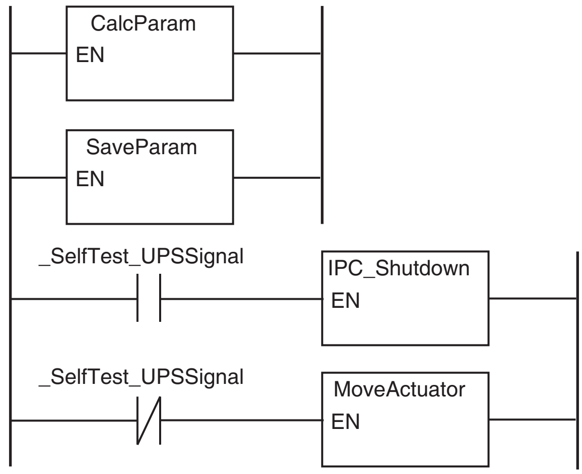
This sample calculates parameters every period and operates the actuator accordingly.

When the temporary power interruption signal is received from the UPS, the instruction saves the calculated parameters and shuts down the operating system (Windows).

_SelfTest_UPSSignal in an NC contacts is inserted to prevent the execution of the MoveActuator instruction when the temporary power interruption signal is received from the UPS.

LD

External Variables	Variable	Data type	Constant	Comment
	_SelfTest_UPSSignal	BOOL	✔	UPS signal detection flag



The CalcParam instruction: User POU. Calculates parameters.
The SaveParam instruction: User POU. Saves parameters.
The MoveActuator instruction: Causes the actuator to operate based on the calculated parameters.

ST

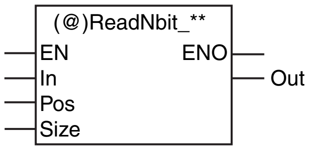
External Variables	Variable	Data type	Constant	Comment
	_SelfTest_UPSSignal	BOOL	✔	UPS signal detection flag

```
CalcParam();
SaveParam();
IF_SelfTest_UPSSignal THEN
  IPC_Shutdown();
ELSE
  MoveActuator();
End_IF;
```

Other Instructions

ReadNbit_**

The ReadNbit_** instructions read zero or more bits from a bit string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ReadNbit_**	N-bit Read Group	FUN	 "**" must be a bit string data type.	Out:=ReadNbit_**(In, Pos, Size); "**" must be a bit string data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Read source	Input	Bit string to read	Depends on data type.	---	0
Pos	Read position		Bit position to read	0 to No. of bits in *In* –1		
Size	Read size		Number of bits to read	0 to No. of bits in *In*		1
Out	Read result	Output	Read result	Depends on data type.	---	---

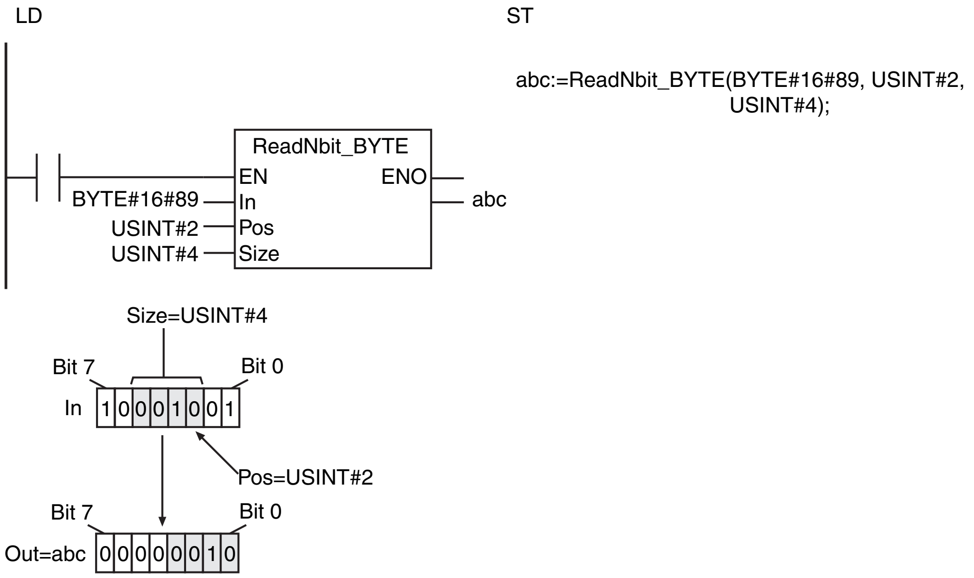
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Pos						OK														
Size						OK														
Out	Must be same data type as *In*																			

Function

A ReadNbit_** instruction reads the values of the upper *Size* bits from read position *Pos* in source bit string *In*. It assigns the values to read result *Out*.

The name of the instruction is determined by the data types of *In* and *Out*. For example, if *In* and *Out* are the WORD data type, the instruction is ReadNbit_WORD.

The following example shows the ReadNbit_BYTE instruction when *In* is BYTE#16#89, *Pos* is USINT#2 and *Size* is USINT#4.



Additional Information

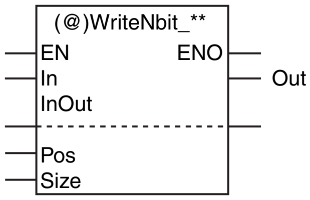
Use a WriteNbit_** instruction to write zero or more bits to a bit string.

Precautions for Correct Use

- The data types of *In* and *Out* must be the same.
- If the value of *Size* is 0, the value of *Out* is 16#0.
- An error occurs in the following cases. *ENO* will be FALSE, and *Out* will not change.
  - The value of *Size* is outside of the valid range.
  - The value of *Pos* is outside of the valid range.
  - The bit string in *In* does not have enough bits for the number of bits specified by *Size* from the position specified by *Pos*.

WriteNbit_**

The WriteNbit_** instructions write zero or more bits to a bit string.

Instruction	Name	FB/FUN	Graphic expression	ST expression
WriteNbit_**	N-bit Write Group	FUN	 "**" must be a bit string data type.	WriteNbit_**(In, Pos, Size, InOut); "**" must be a bit string data type.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Read source	Input	Bit string from which to read bits to write to *InOut*	Depends on data type.	---	0
Pos	Write position		Bit position to which to write	0 to No. of bits in *InOut* –1		
Size	Write size		Number of bits to write	0 to No. of bits in *In*		1
InOut	Write target	In-out	Write result	Depends on data type.	---	---
Out	Return value	Output	Always TRUE	TRUE only	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		OK	OK	OK	OK															
Pos						OK														
Size						OK														
InOut	Must be same data type as *In*																			
Out	OK																			

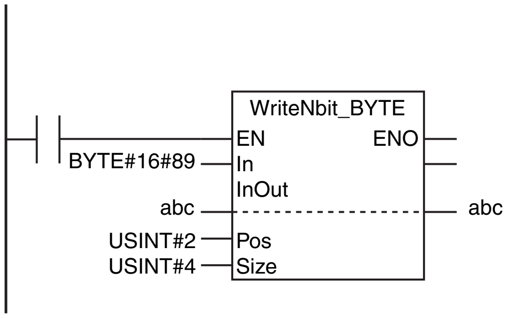
Function

A WriteNbit_** instruction first reads the lower *Size* bits from read source *In*. Then it writes the values that it read to write position *Pos* in write target *InOut*.

The name of the instruction is determined by the data types of *In* and *Out*. For example, if *In* and *Out* are the WORD data type, the instruction is WriteNbit_WORD.

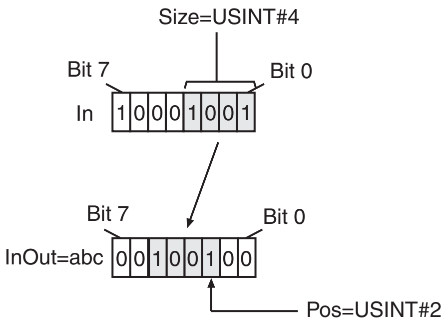
The following example shows the WriteNbit_BYTE instruction when *In* is BYTE#16#89, *Pos* is USINT#2 and *Size* is USINT#4.

LD



ST

```
WriteNbit_BYTE(BYTE#16#89, abc,
                    USINT#2, USINT#4);
```



Additional Information

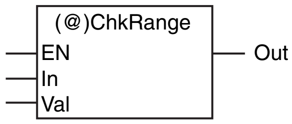
Use a ReadNbit_** instruction to read zero or more bits from a bit string.

Precautions for Correct Use

- The data types of *In* and *InOut* must be the same.
- The value of *InOut* does not change if the value of *Size* is 0.
- Return value *Out* is not used when the instruction is used in ST.
- An error occurs in the following cases. *ENO* will be FALSE, and *InOut* will not change.
  - The value of *Size* is outside of the valid range.
  - The value of *Pos* is outside of the valid range.
  - The bit string in *InOut* does not have enough bits for the number of bits specified by *Size* from the position specified by *Pos*.

ChkRange

The ChkRange instruction determines if the value of a variable is within the valid range of the range type specification.

Instruction	Name	FB/FUN	Graphic expression	ST expression
ChkRange	Check Subrange Variable	FUN		Out:=ChkRange(In, Val);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
In	Variable to check	Input	Variable to check	Depends on data type.	---	*
Val	Range specification variable		Range specification variable	Depends on the range specification.		
Out	Check result	Output	Check result	Depends on data type.	---	---

* If you omit the input parameter, the default value is not applied. A building error will occur.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						OK	OK	OK	OK	OK	OK	OK	OK							
Val	The basic data type that is the basis for the range specification must be the same as *In*.																			
Out	OK																			

Function

The ChkRange instruction determines if the value of variable to check *In* is within the valid range of the range specification variable *Val*. If the value is within the valid range, check result *Out* is TRUE. If the value is not within the valid range, check result *Out* is FALSE.

Additional Information

You can define the range type specification for integer variables (USINT, UINT, UDINT, ULINT, SINT, INT, DINT, and LINT).

Precautions for Correct Use

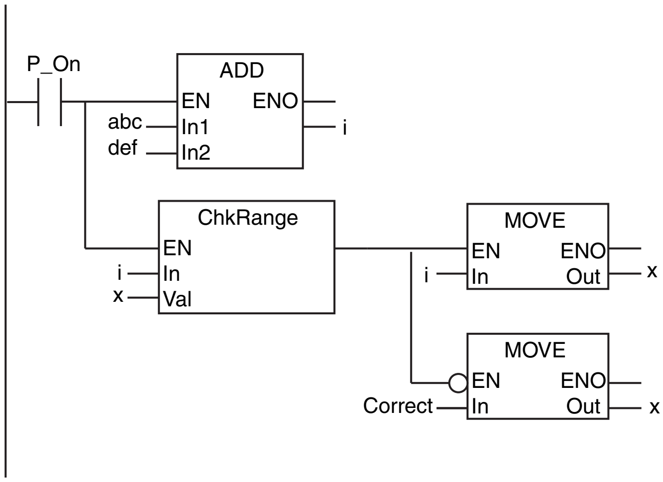
- If *In* is not a range specification variable, the value of *Out* changes to TRUE.
- If this instruction is used in a ladder diagram, the value of *Out* changes to FALSE if an error occurs in the previous instruction on the rung.

Sample Programming

Here, the result of addition *i* is checked to see if it is within the valid range (10 to 99) of the range specification variable *x*. If it is not within the valid range, the value of variable *Correct* is assigned to variable *x*.

LD

Variable	Data type	Initial value
i	INT	0
abc	INT	0
def	INT	0
x	INT(10..99)	10
Correct	INT	0



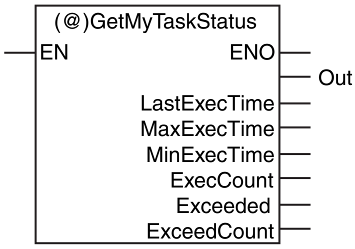
ST

Variable	Data type	Initial value
i	INT	0
abc	INT	0
def	INT	0
Chk	BOOL	FALSE
x	INT(10..99)	10
Correct	INT	0

```
i := abc+def;
Chk:=ChkRange(i, x);    // Check subrange variable.

IF (Chk=TRUE) THEN
     x := i;            // Assign i to x if value of i is in range.
ELSE
     x := Correct;      // Assign Correct to x if value of i is out of range.
END_IF;
```

GetMyTaskStatus

The GetMyTaskStatus reads the status of the current task.

Instruction	Name	FB/FUN	Graphic expression	ST expression
GetMyTaskStatus	Read Current Task Status	FUN		GetMyTaskStatus(LastExecTime, MaxExecTime, MinExecTime, ExecCount, Exceeded, ExceedCount);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Out	Return value	Output	Always TRUE	TRUE only	---	---
LastExec Time	Last task execution time		Last task execution time of the current task	Depends on data type.*	ns	
MaxExec Time	Maximum task execution time		Maximum task execution time of the current task			
MinExec Time	Minimum task execution time		Minimum task execution time of the current task			
ExecCount	Task execu-tion count		Number of task executions of the current task	Depends on data type.	---	
Exceeded	Task period exceeded flag		TRUE: The last execution of the current task was not completed within the task period. FALSE: The last execution of the current task was com-pleted within the task period.			
Exceed-Count	Task period exceeded count		The number of times the current task has exceeded the task period.			

* Negative numbers are excluded.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out						OK														
LastExec Time																OK				
MaxExec Time																OK				
MinExec Time																OK				

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
ExecCount								OK												
Exceeded	OK																			
Exceed-Count								OK												

Function

The GetMyTaskStatus reads the status of the current task. The task status includes the last task execution time *LastExecTime*, maximum task execution time *MaxExecTime*, minimum task execution time *MinExecTime*, task execution count *ExecCount*, task period exceeded flag *Exceeded*, and task period exceeded count *ExceedCount*.

Additional Information

MaxExecTime, *MinExecTime*, *ExecCount*, and *ExceedCount* are reset at the following times.

- When operation is started
- When a reset operation is executed from the Task Execution Time Monitor of the Sysmac Studio.

Precautions for Correct Use

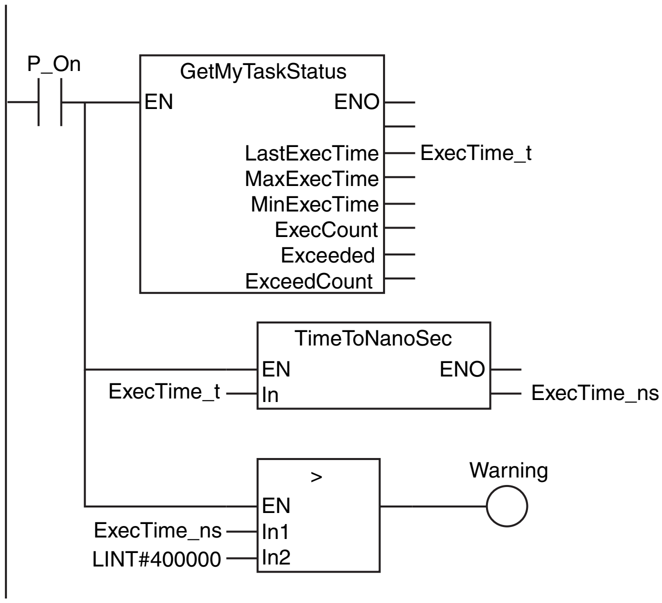
- When the value of *ExecCount* or *ExceedCount* exceeds the maximum value of UDINT data (4,294,967,295), it returns to 0.
- Return value *Out* is not used when the instruction is used in ST.

Sample Programming

In this sample, the GetMyTaskStatus reads the status of the current task. If the previous task execution time exceeds 400 μs (400000 ns), the value of the *Warning* variable changes to TRUE.

LD

Variable	Data type	Initial value	Comment
ExecTime_t	TIME	T#0s	Previous task execution time (TIME data)
ExecTime_ns	INT	0	Previous task execution time (nanoseconds LINT data)
Warning	BOOL	FALSE	Warning



ST

Variable	Data type	Initial value	Comment
ExecTime_t	TIME	T#0s	Previous task execution time (TIME data)
ExecTime_ns	LINT	0	Previous task execution time (nanoseconds LINT data)
Warning	BOOL	FALSE	Warning

```
GetMyTaskStatus(LastExecTime=>ExecTime_t); // Get previous task period.
ExecTime_ns:=TimeToNanoSec(ExecTime_t);    // Convert previous task period from TIME data to nanoseconds.
IF (ExecTime_ns>DINT#400000) THEN          // If previous task period exceeds 400,000 ns...
     Warning:=TRUE;                        //  Assign TRUE to Warning variable.
ELSE
     Warning:=FALSE;
END_IF;
```

GetMyTaskInterval

The GetMyTaskInterval instruction reads the task period of the current task.

Instruction	Name	FB/FUN	Graphic expression	ST expression
GetMyTask Interval	Read Current Task Period	FUN		Out:=GetMyTaskInterval();

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Out	Task period	Output	Task period of current task	Depends on data type.*1	ms	---

*1 Negative numbers are excluded.

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out																OK				

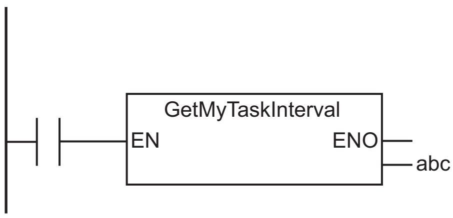
Function

The GetMyTaskInterval instruction reads the task period of the current task and stores it in task period *Out* if the task that executes the instruction is the primary periodic task or a periodic task.

If an event task executes the instruction, the value of *Out* will be T#0s.

The following figure shows a programming example. If the task period of the current task is 1 ms, the value of *abc* will be T#1ms.

LD



ST

abc:=GetMyTaskInterval();

Version Information

A CPU Unit with unit version 1.08 or later and Sysmac Studio version 1.09 or higher are required to use this instruction.

Sample Programming

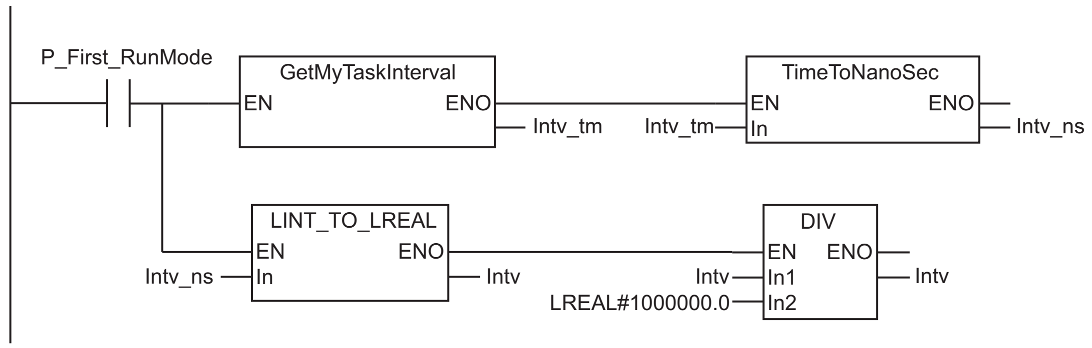
This example reads the task period of the current task when this program is first executed after operation starts. Then the task period that was read is converted from TIME data to LREAL data in milliseconds. This sample programming can be used, for example, to calculate the axis target position for each task period.

The following procedure is used to convert TIME data to LREAL data in milliseconds.

1. The GetMyTaskInterval instruction is used to read the task period as TIME data.
2. The TimeToNanoSec instruction is used to convert TIME data to LINT data in nanoseconds.
3. The LINT_TO_LREAL instruction is used to convert LINT data in nanoseconds to LREAL data in nanoseconds.
4. The DIV instruction is used to divide the result of step 3 by 1,000,000 to convert to milliseconds.

LD

Variable	Data type	Default	Comment
Intv_tm	TIME	T#0s	Task period as TIME data
Intv_ns	LINT	0	Task period as LINT data in nanoseconds
Intv	LREAL	0	Task period as LREAL data in milliseconds



ST

Variable	Data type	Default	Comment
Intv	LREAL	0	Task period as LREAL data in milliseconds

```
IF P_First_RunMode = TRUE THEN
   Intv := LINT_TO_LREAL(TimeToNanoSec(GetMyTaskInterval()))/1000000;
END_IF;
```

Task_IsActive

The Task_IsActive instruction determines if the specified task is currently in execution.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Task_IsActive	Determine Task Status	FUN	(@)Task_IsActive EN — Out TaskName	Out:=Task_IsActive(TaskName);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
TaskName	Task name	Input	Task name	64 bytes max. (63 single-byte alphanumeric characters plus the final NULL character)	---	''
Out	Judgement	Output	TRUE: Task is in execution or on standby. FALSE: Not active	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
TaskName																				OK
Out	OK																			

Function

The Task_IsActive instruction determines if the task specified with *TaskName* is currently in execution or on standby. “On standby” means that a high-priority task was started after this task was started, so processing has been interrupted.
If it is being executed or on standby, the value of judgment *Out* is TRUE. If it is not being executed, the value of *Out* is FALSE.

Precautions for Correct Use

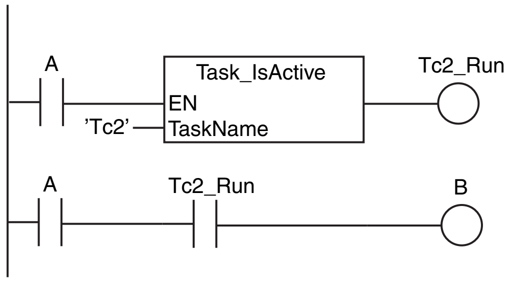
- You cannot use a variable to which a text string was assigned for *TaskName*. Directly specify a text string.
- If this instruction is used in a ladder diagram, the value of *Out* changes to FALSE if an error occurs in the previous instruction on the rung.
- An error occurs in the following case. The value of *Out* does not change.
  - The task specified with *TaskName* does not exist.

Sample Programming

In this sample, the instruction determines whether periodic task Tc2 is active when the value of variable *A* changes to TRUE. If it is active, the value of variable *B* changes to TRUE.

LD

Variable	Data type	Initial value	Comment
A	BOOL	FALSE	
B	BOOL	FALSE	
Tc2_Run	BOOL	FALSE	Task Tc2 execution status



ST

Variable	Data type	Initial value	Comment
A	BOOL	FALSE	
B	BOOL	FALSE	
Tc2_Run	BOOL	FALSE	Task Tc2 execution status

```
IF (A=TRUE) THEN
     // Determine task status.
     Tc2_Run:=Task_IsActive('Tc2');
     // Make variable B TRUE if Tc2 is running.
     IF (Tc2_Run=TRUE) THEN
          B := TRUE;
     END_IF;

END_IF;
```

Lock and Unlock

Lock: Starts an exclusive lock between tasks. Execution of any other task with a lock region with the same lock number is disabled.

Unlock: Stops an exclusive lock between tasks.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Lock	Lock Tasks	FUN	(@)Lock EN ENO Index Out	Lock(Index);
Unlock	Unlock Tasks	FUN	(@)Unlock EN ENO Index Out	Unlock(Index);

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Index	Lock number	Input	Lock number	Depends on data type.	---	0
Out	Return value	Output	Always TRUE	TRUE only	---	---

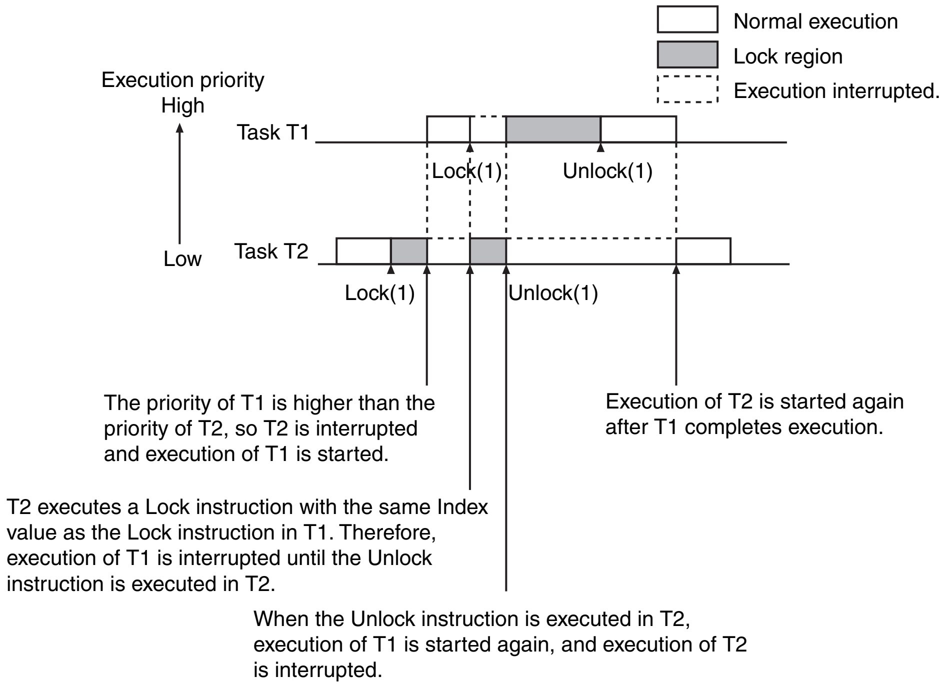
	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Index						OK														
Out	OK																			

Function

The Lock and Unlock instructions create lock regions. If a lock region in one task is being executed, the lock regions with the same lock number in other tasks are not executed. Specify the lock number with *Index*.

The following figure shows a programming example.

Both task T1 and task T2 contain a lock region with *Index* set to 1. If the Lock instruction in T2 is executed first, the lock region in T1 is not executed until the Unlock instruction is executed in T2.



Lock regions with different values for *Index* do not affect each other.

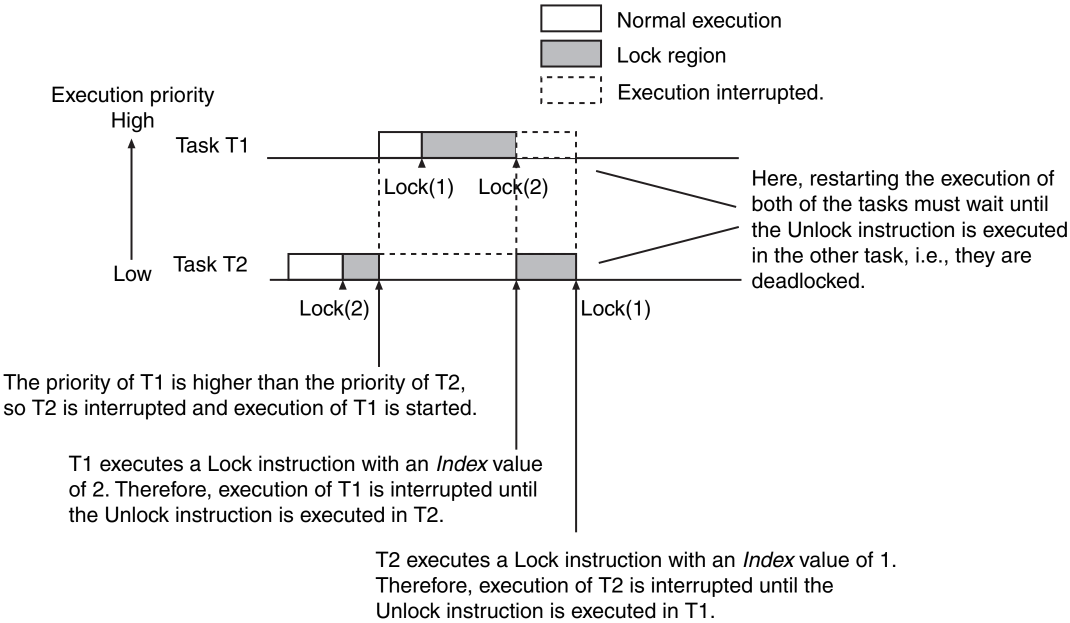
Additional Information

- The Lock and Unlock instructions are used when the same data is read/written from more than one task. They are used to prevent other tasks from reading/writing the data while a certain task is reading/writing the data.
- As long as the *Index* values are different, more than one pair of Lock and Unlock instructions can be placed in the same POU. The instruction pairs can also be nested.

Precautions for Correct Use

- Do not make lock regions any longer than necessary. If the lock region is too long, the task execution period may be exceeded.
- Always use the Lock and Unlock instructions together as a set in the same section of the same POU.
- You can set a maximum of 16,777,215 lock regions at the same time.

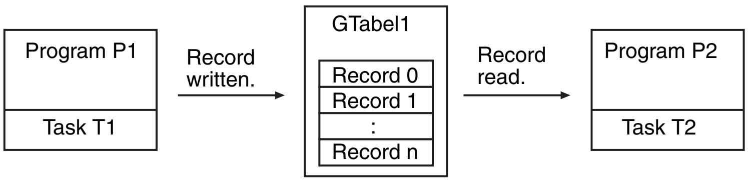
- If Lock instructions are used in more than one task, a deadlock may occur if they are positioned poorly. A Task Execution Timeout Error will occur if there is a deadlock and a total stop is performed.



- An error occurs in the following case. The value of *Out* does not change.
  - There are more than 16,777,215 lock region at the same time.

Sample Programming

Here, program P1 in task T1 and program P2 in task T2 both access the same global variable *GTable1*. When the value of write request *WriteReq* changes to TRUE, P1 writes one record to record array *GTable1.Record[]* and increments *GTable1.Index*. When read request *ReadReq* changes to TRUE, P2 decrements *GTable1.Index* and reads one record from *GTable1.Record[]*.
The Lock instruction is used so that reading and writing do not occur at the same time.



Definition of Global Variable *GTable*

Data type

Variable	Data type	Comment
USERTABLE	STRUCT	Record storage structure
Index	INT	Index
Record	ARRAY[0..99] OF LREAL	Record array

Global Variables

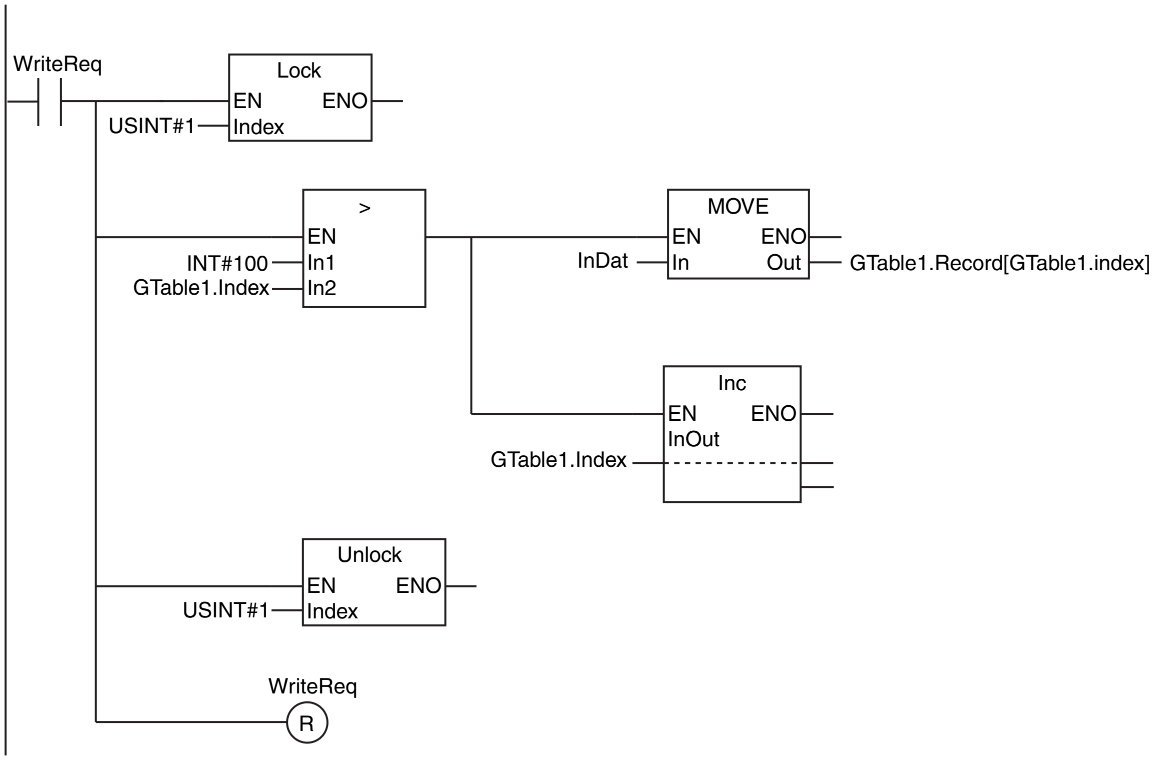
Variable	Data type	Initial value	Comment
GTable1	USERTABLE	(Index:=0,Record:=[100(0.0)])	Record storage structure

Program P1

LD

Internal Variables	Variable	Data type	Initial value	Comment
	WriteReq	BOOL	FALSE	Write request
	InDat	LREAL	0.0	Write data

External Variables	Variable	Data type	Comment
	GTable1	USERTABLE	Record storage structure



ST

Internal Variables	Variable	Data type	Initial value	Comment
	WriteReq	BOOL	FALSE	Write request
	InDat	LREAL	0.0	Write data

External Variables	Variable	Data type	Comment
	GTable1	USERTABLE	Record storage structure

```
// Detect write request.
IF (WriteReq=TRUE) THEN

     // Execute Lock instruction.
     Lock(USINT#1);

     IF (INT#100>GTable1.Index) THEN
          GTable1.Record[GTable1.Index]:=InDat;
          GTable1.Index                :=GTable1.Index+INT#1;
     END_IF;

     // Execute Unlock instruction.
```

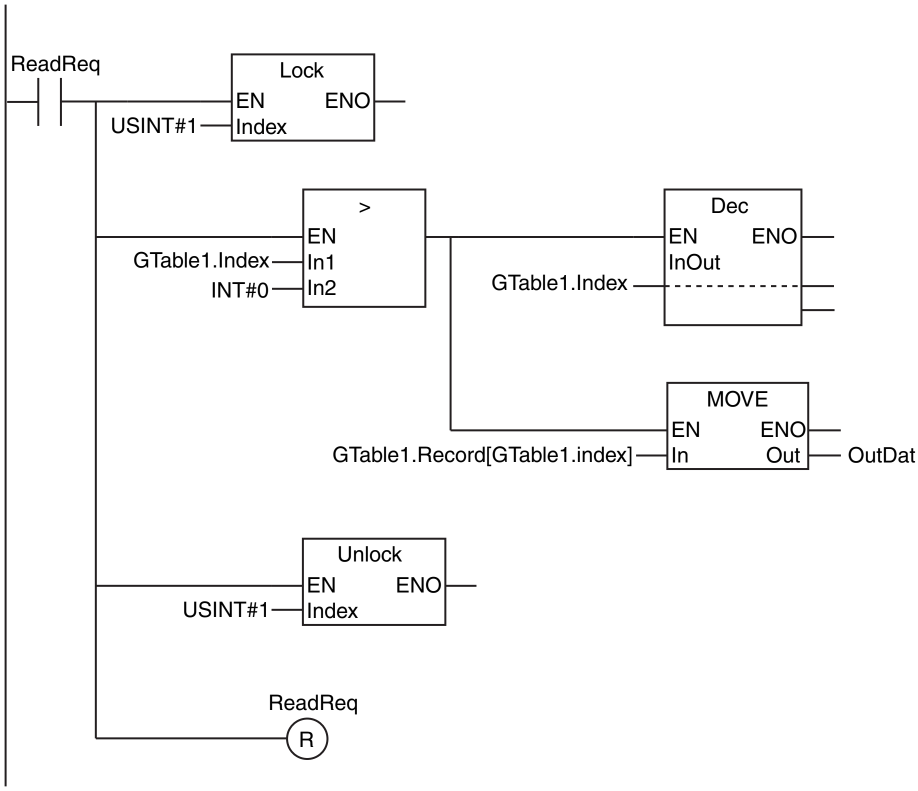
```
        Unlock(USINT#1);
        WriteReq:=FALSE;

    END_IF;
```

Program P2

LD

Internal Variables	Variable	Data type	Initial value	Comment
	ReadReq	BOOL	FALSE	Read request
	OutDat	LREAL	0.0	Read data

External Variables	Variable	Data type	Comment
	GTable1	USERTABLE	Record storage structure



ST

Internal Variables	Variable	Data type	Initial value	Comment
	ReadReq	BOOL	FALSE	Read request
	OutDat	LREAL	0.0	Read data

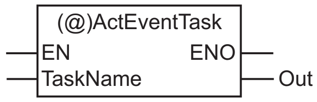
External Variables	Variable	Data type	Comment
	GTable1	USERTABLE	Record storage structure

```
// Detect read request.
IF (ReadReq=TRUE) THEN

     // Execute Lock instruction.
     Lock(USINT#1);

     IF (GTable1.Index>INT#0) THEN
          GTable1.Index:=GTable1.Index-INT#1;
          OutDat       :=GTable1.Record[GTable1.Index];
     END_IF;
```

```
        // Execute Unlock instruction.
        Unlock(USINT#1);
        ReadReq:=FALSE;

END_IF;
```

ActEventTask

The ActEventTask instruction activates an event task.

Instruction	Name	FB/ FUN	Graphic expression	ST expression
ActEventTask	Activate Event Task	FUN		ActEventTask(TaskName);

Variables

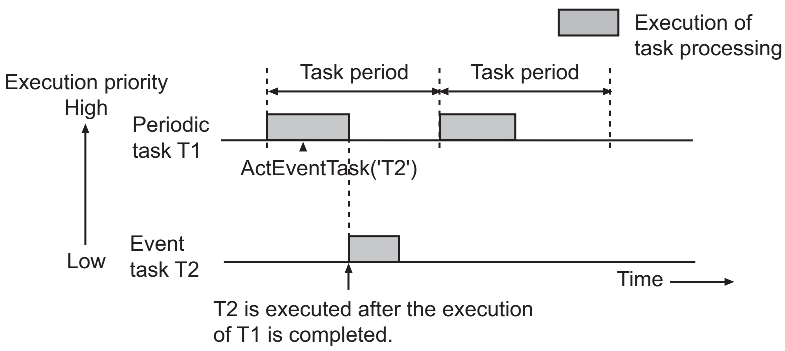
Name	Meaning	I/O	Description	Valid range	Unit	Default
TaskName	Task name	Input	The name of the event task to activate	64 bytes max. (63 single-byte alphanumeric characters plus the final NULL character)	---	''
Out	Return value	Output	TRUE: The instruction was executed without any errors. FALSE: The instruction was not executed or an error occurred.	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
TaskName																				OK
Out	OK																			

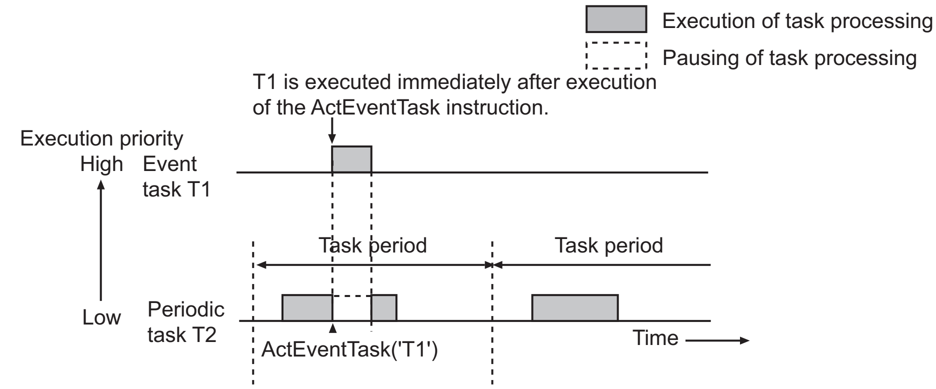
Function

The ActEventTask instruction activates the event task with task name *TaskName*. The event task operates according to its task execution priority.

If an event task is started that has an execution priority that is lower than the execution priority of the task in which this instruction was executed, the event task is executed after completion of the execution of the task in which this instruction was executed. For example, assume that the execution priority of event task T2 is lower than the execution priority of periodic task T1. If the ActEventTask instruction is executed for T2 in T1, the execution of T1 is completed before T2 is executed.

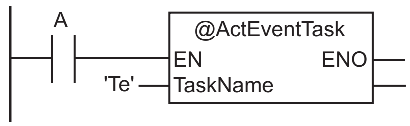


If an event task is started that has an execution priority that is higher than the execution priority of the task in which this instruction was executed, the execution of the task in which this instruction was executed is paused and the event task is executed. For example, assume that the execution priority of periodic task T2 is lower than the execution priority of event task T1. If the ActEventTask instruction is executed for T1 in T2, the execution of T2 is paused to execute T1.



The following figure shows a programming example. When the value of variable *A* is TRUE, event task 'Te' is executed.

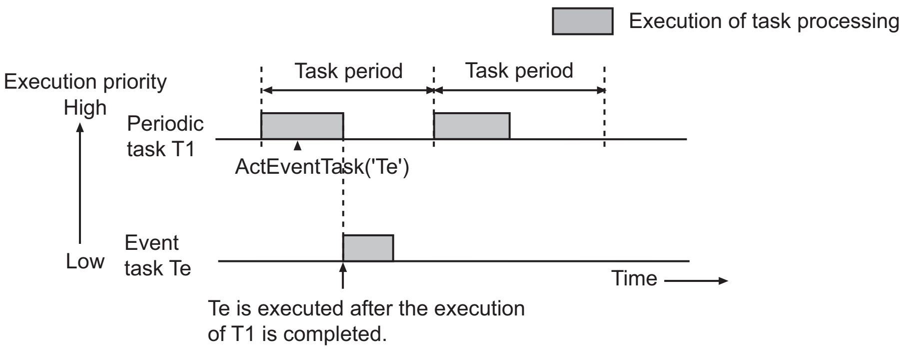
LD



ST

```
IF A=TRUE THEN
    ActEventTask('Te');
END_IF;
```

Assume that the program with these instructions is assigned to periodic task T1 and that the execution priority of Te is lower than that of T1. If this instruction is executed in T1, the execution of T1 is completed before Te is executed.



Related System-defined Variables

Name	Meaning	Data type	Description
_**_Active*1	Task Active Flag	BOOL	This variable indicates the execution status of the task.*2 TRUE: Execution processing is in progress. FALSE: Stopped.

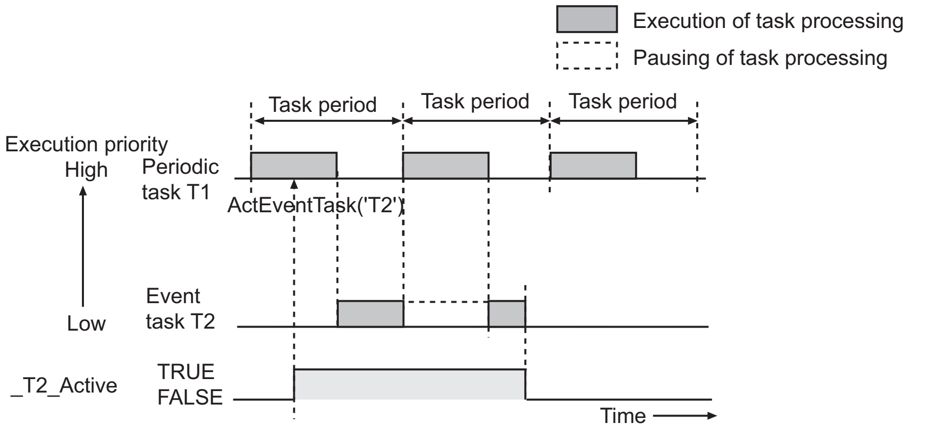
*1 The asterisks (**) are replaced with the task name.

*2 Refer to the *NJ/NX-series CPU Unit Software User's Manual* (Cat. No. W501) or *NY-series Industrial Panel PC / Industrial Box PC Software User's Manual* (Cat. No. W558) for details.

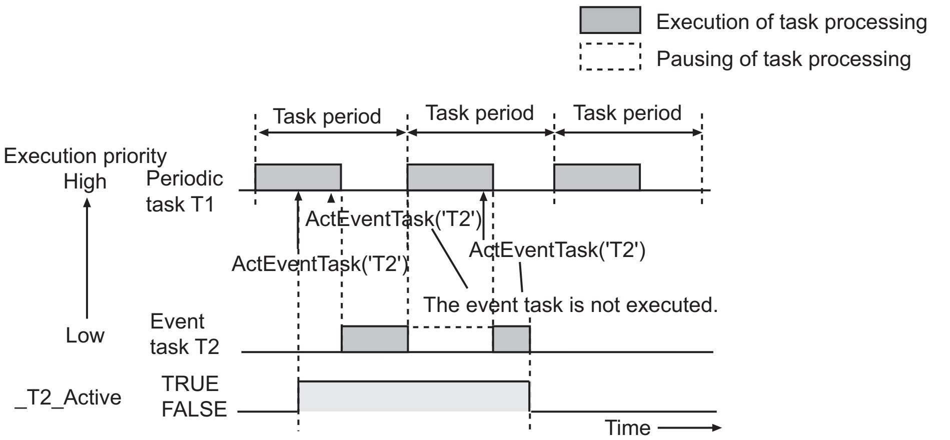
Additional Information

Operation of _**_*Active* System-defined Variable

- When this instruction is executed, the _**_*Active* system-defined variable for the specified event task will change to TRUE. It will change to FALSE when execution of the event task is completed. For example, assume that the execution priority of event task T2 is lower than the execution priority of periodic task T1. When the ActEventTask instruction is executed for T2 in T1, the system-defined variable *_T2_Active* will change as shown in the following figure.



- The event task will not be executed even if this instruction is executed while the system-defined variable _**_*Active* for the event task is TRUE.

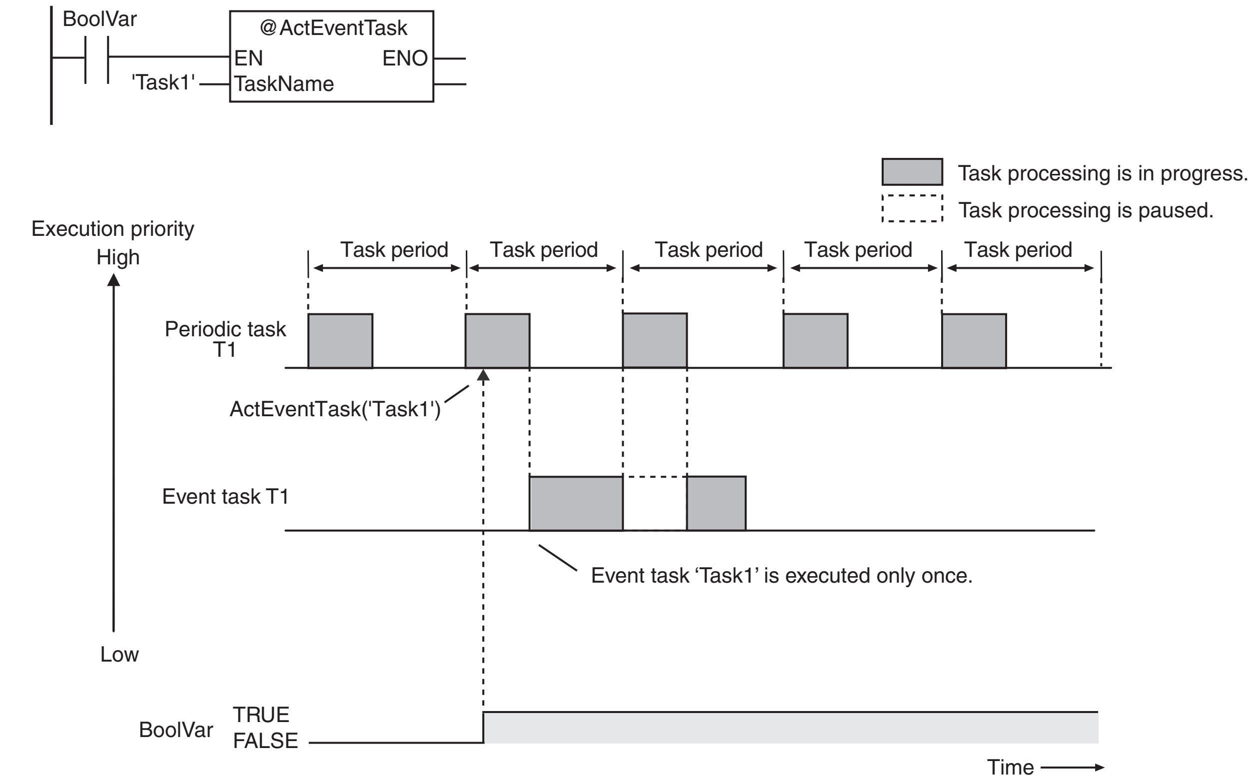


Executing an Event Task Only Once and Executing It Repeatedly

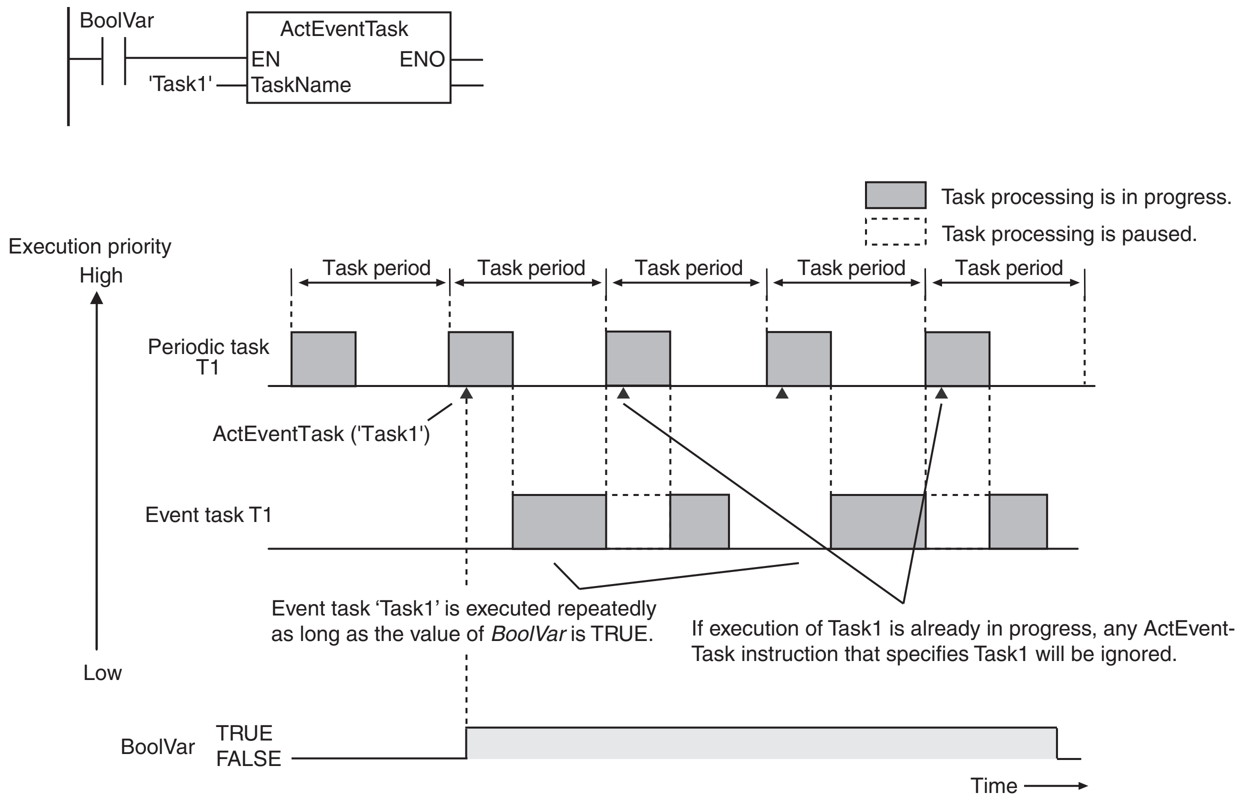
Use the following type of programming when you want to execute an event task only once when the value of a specified variable changes and when you want to execute an event task repeatedly as long as the variable has a specific value.

Example 1: Executing an Event Task Only Once When the Value of a Specified Variable Changes

If you use an upward differentiation instruction option for the instruction as shown below, event task ‘Task1’ will be executed only once when the value of BOOL variable *BoolVar* changes from FALSE to TRUE.



Example 2: Executing an Event Task Repeatedly as Long as a Variable Has a Specific Value
If you do not use an upward differentiation instruction option for the instruction as shown below, event task 'Task1' will be executed repeatedly as long as the value of BOOL variable *BoolVar* is TRUE. However, if this instruction is executed for Task1 while Task1 execution is in progress, it will be ignored.



Precautions for Correct Use

- To reduce the instruction execution time, execute this instruction only when it is necessary to execute the event task. If the instruction is executed while the system-defined variable _**_*Active* is TRUE, processing time is required even if the event task is not executed.
- An error will occur if the event task that is specified with *TaskName* does not exist. *ENO* will be FALSE.

Version Information

A CPU Unit with unit version 1.03 or later and Sysmac Studio version 1.04 or higher are required to use this instruction.

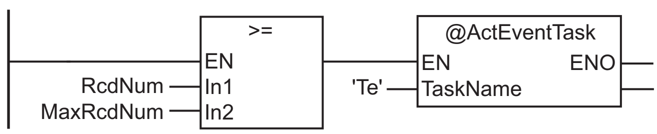
Sample Programming

● Example of Executing an Event Task When the Value of a Variable Meets the Specified Condition

Event task 'Te' is executed only once when the value of variable *RcdNum* changes from less than the value of the variable *MaxRcdNum* to greater than or equal to the value of *MaxRcdNum.*

LD

Variable	Data type	Initial value
RcdNum	INT	0
MaxRcdNum	INT	100



ST

Variable	Data type	Initial value
RcdNum	INT	0
MaxRcdNum	INT	100
met	BOOL	FALSE

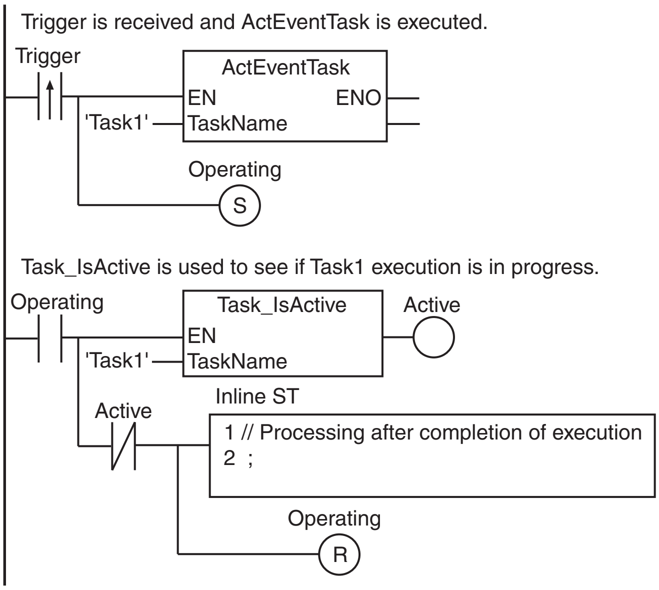
```
IF (RcdNum>=MaxRcdNum) THEN
   IF (met=FALSE) THEN
      ActEventTask('Te');
      met:=TRUE;
   END_IF;
ELSE
   met:=FALSE;
END_IF;
```

● Example of Confirming Completion of Event Task before Proceeding

In this example, event task 'Task1' is executed each time the value of *Trigger* changes to TRUE. The Task_IsActive instruction is used to see when execution of Task 1 is completed.

LD

Name	Data type	Initial value	Comment
Trigger	BOOL	FALSE	Execution condition
Operating	BOOL	FALSE	Checking event task execution in progress
Active	BOOL	FALSE	Event task execution in progress



ST

Name	Data type	Initial value	Comment
Trigger	BOOL	FALSE	Execution condition
LastTrigger	BOOL	FALSE	Value of *Trigger* from previous task period
Operating	BOOL	FALSE	Checking event task execution in progress
Active	BOOL	FALSE	Event task execution in progress

```
// Start sequence when Trigger changes to TRUE.
IF ( (Trigger=TRUE) AND (LastTrigger=FALSE) ) THEN
   ActEventTask('Task1');        // Execute event task 'Task1'.
   Operating:=TRUE;
END_IF;
LastTrigger:=Trigger;

// See if Task1 execution is in progress.
IF (Operating=TRUE) THEN
   Active:=Task_IsActive('Task1');

   IF (Active=FALSE) THEN        // Task1 execution completed.
      Operating:=FALSE;
   END_IF;

END_IF;
```

Get**Clk

The Get**Clk instruction outputs a clock pulse at the specified cycle.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Get**Clk	Get Clock Pulse Group	FUN	 "**" must be 100 us, 1 ms, 10 ms, 20 ms, 100 ms, 1 s, or 1 min.	Out:=Get**Clk(); “**” must be 100 us, 1 ms, 10 ms, 20 ms, 100 ms, 1 s, or 1 min.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Out	Clock pulse	Output	Clock pulse	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out	OK																			

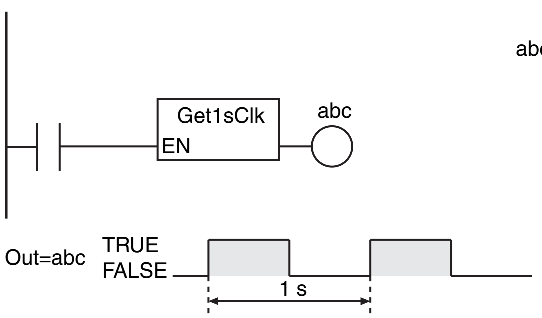
Function

The Get**Clk instruction outputs a clock pulse at the specified cycle.
The clock pulse period is 100 us, 1 ms, 10 ms, 20 ms, 100 ms, 1 s, or 1 min.

The name of the instruction is determined by the period of the clock pulse. For example, if the period of the clock pulse is 10 ms, the instruction name is Get10msClk.

The following example is for the Get1sClk instruction.

LD



ST

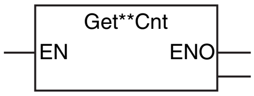
```
abc:=Get1sClk();
```

Precautions for Correct Use

- When the instruction is executed, the first value of *Out* may be TRUE or it may be FALSE.
- If this instruction is used in a ladder diagram, the value of *Out* changes to FALSE if an error occurs in the previous instruction on the rung.

Get**Cnt

The Get**Cnt instruction gets the values of free-running counters of the specified cycle.

Instruction	Name	FB/FUN	Graphic expression	ST expression
Get**Cnt	Get Incrementing Free-running Counter Group	FUN	 "**" must be 100 ns, 1us, 1 ms, 10 ms, 100 ms, or 1 s.	Out:=Get**Cnt(); "**" must be 100 ns, 1 us, 1 ms, 10 ms, 100 ms, or 1 s.

Variables

Name	Meaning	I/O	Description	Valid range	Unit	Default
Out	Count	Output	Value of free-running counter	Depends on data type.	---	---

	Boolean	Bit strings				Integers								Real numbers		Times, durations, dates, and text strings				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out									OK											

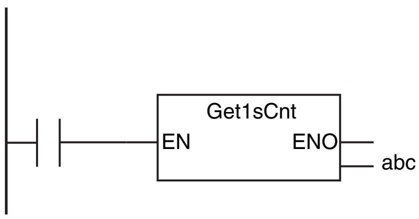
Function

The Get**Cnt instruction gets the values of free-running counters of the specified cycle.
A free-running counter is a counter that is incremented at a specific period. *Out* is the current value of the count. The counter period is 100 ns, 1 us, 1 ms, 10 ms, 100 ms, or 1 s.

The name of the instruction is determined by counter period. For example, if the counter period is 10 ms, the instruction name is Get10msCnt.

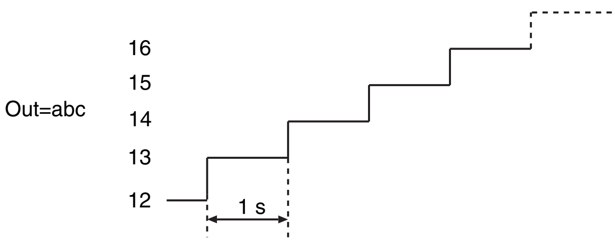
The following example is for the Get1sCnt instruction.

LD



ST

```
abc:=Get1sCnt();
```



Precautions for Correct Use

- Free-running counters start counting as soon as the power supply is turned ON. When the count exceeds the valid range of ULINT data (18,446,744,073,709,551,615), it returns to 0 and counting continues.
- This instruction only gets the current value of the free-running counter. It does not reset the counter to 0.
- The first value of *Out* cannot be predicted. It will not necessarily start from 0.

A

Appendices

A-1 Error Codes That You Can Check with *ErrorID*

Error codes are assigned to the errors that can occur when instructions are executed. When you use instructions that have an error code output variable (*ErrorID*), you can use the error codes to program error processing.

The following table lists the instructions with *ErrorID* and the error codes that can occur for those instructions. Refer to the *NY-series Troubleshooting Manual* (Cat. No. W564) for the meanings of the error codes.

Additional Information

You can check for errors for instructions that do not have *ErrorID* in the events in the event log.

Type	Instruction	Name	Error code	Error name
Analog Control Instructions	PIDAT	PID Control with Autotuning	16#0400	Input Value Out of Range
			16#0401	Input Mismatch
	PIDAT_HeatCool	Heating/Cooling PID with Autotuning	16#0400	Input Value Out of Range
			16#0401	Input Mismatch
	AC_StepProgram	Step Program	16#0400	Input Value Out of Range
System Control Instructions	ResetPLCError	Reset PLC Controller Error	---	---
	ResetMCError	Reset Motion Control Error	---	---
	ResetECError	Reset EtherCAT Error	16#041A	Multi-execution of Instructions
	RestartNXUnit	Restart NX Unit	16#0400	Input Value Out of Range
			16#0419	Incorrect Data Type
			16#2C00	NX Message Error
			16#2C01	NX Message Resource Overflow
			16#2C02	NX Message Timeout
			16#2C05	NX Message EtherCAT Network Error
			16#2C06	External Restart Already Executed for Specified NX Units
			16#2C07	Unapplicable Unit Specified for Instruction
	NX_ChangeWriteMode	Change to NX Unit Write Mode	16#0400	Input Value Out of Range
			16#0419	Incorrect Data Type
			16#2C00	NX Message Error
			16#2C01	NX Message Resource Overflow
			16#2C02	NX Message Timeout
			16#2C05	NX Message EtherCAT Network Error
			16#2C07	Unapplicable Unit Specified for Instruction

Type	Instruction	Name	Error code	Error name
System Control Instructions	NX_SaveParam	Save NX Unit Parameters	16#0400	Input Value Out of Range
			16#0419	Incorrect Data Type
			16#2C00	NX Message Error
			16#2C01	NX Message Resource Overflow
			16#2C02	NX Message Timeout
	NX_ReadTotalPowerOnTime	Read NX Unit Total Power ON Time	16#0400	Input Value Out of Range
			16#0419	Incorrect Data Type
			16#2C00	NX Message Error
			16#2C01	NX Message Resource Overflow
			16#2C02	NX Message Timeout
			16#2C08	Invalid Total Power ON Time Record
EtherCAT Communications Instructions	EC_CoESDOWrite	Write EtherCAT CoE SDO	16#0400	Input Value Out of Range
			16#1800	EtherCAT Communications Error
			16#1801	EtherCAT Slave Does Not Respond
			16#1802	EtherCAT Timeout
			16#1804	SDO Abort Error
			16#1808	Communications Resource Overflow
	EC_CoESDORead	Read EtherCAT CoE SDO	16#0400	Input Value Out of Range
			16#1800	EtherCAT Communications Error
			16#1801	EtherCAT Slave Does Not Respond
			16#1802	EtherCAT Timeout
			16#1803	Reception Buffer Overflow
			16#1804	SDO Abort Error
			16#1808	Communications Resource Overflow
	EC_StartMon	Start EtherCAT Packet Monitor	16#1805	Saving Packet Monitor File
			16#1807	Packet Monitoring Function in Operation
			16#1808	Communications Resource Overflow
	EC_StopMon	Stop EtherCAT Packet Monitor	16#1806	Packet Monitoring Function Not Started
			16#1808	Communications Resource Overflow
	EC_SaveMon	Save EtherCAT Packets	16#1805	Saving Packet Monitor File
			16#1807	Packet Monitoring Function in Operation
			16#1808	Communications Resource Overflow

Type	Instruction	Name	Error code	Error name
EtherCAT Communications Instructions	EC_CopyMon	Transfer EtherCAT Packets	16#0400	Input Value Out of Range
			16#1400	SD Memory Card Access Failure
			16#1401	SD Memory Card Write-protected
			16#1402	SD Memory Card Insufficient Capacity
			16#1403	File Does Not Exist
			16#1404	Too Many Files/ Directories
			16#1405	File Already in Use
			16#140A	Write Access Denied
			16#140B	Too Many Files Open
			16#140D	File or Directory Name Is Too Long
			16#140E	SD Memory Card Access Failed
			16#1808	Communications Resource Overflow
	EC_DisconnectSlave	Disconnect EtherCAT Slave	16#1800	EtherCAT Communications Error
			16#1801	EtherCAT Slave Does Not Respond
			16#1808	Communications Resource Overflow
	EC_ConnectSlave	Connect EtherCAT Slave	16#1800	EtherCAT Communications Error
			16#1801	EtherCAT Slave Does Not Respond
			16#1808	Communications Resource Overflow
	EC_ChangeEnableSetting	Enable/Disable EtherCAT Slave	16#1800	EtherCAT Communications Error
			16#1801	EtherCAT Slave Does Not Respond
			16#1808	Communications Resource Overflow
	NX_WriteObj	Write NX Unit Object	16#0400	Input Value Out of Range
			16#0419	Incorrect Data Type
			16#041B	Data Capacity Exceeded
			16#2C00	NX Message Error
			16#2C01	NX Message Resource Overflow
			16#2C02	NX Message Timeout
			16#2C03	Incorrect NX Message Length
	NX_ReadObj	Read NX Unit Object	16#0400	Input Value Out of Range
			16#0410	Text String Format Error
			16#0419	Incorrect Data Type
			16#041C	Different Data Sizes
			16#2C00	NX Message Error
			16#2C01	NX Message Resource Overflow
			16#2C02	NX Message Timeout

Type	Instruction	Name	Error code	Error name
IO-Link Communications Instructions	IOL_ReadObj	Read IO-Link Device Object	16#0400	Input Value Out of Range
			16#0410	Text String Format Error
			16#0419	Incorrect Data Type
			16#041C	Different Data Sizes
			16#4800	Device Error Received
			16#4801	Specified Unit Does Not Exist
			16#4802	Message Processing Limit Exceeded
			16#4803	Specified Unit Status Error
			16#4804	Too Many Simultaneous Instruction Executions
			16#4805	Communications Timeout
			16#4806	Invalid Mode
			16#4807	I/O Power OFF Status
			16#4808	Verification Error
	IOL_WriteObj	Write IO-Link Device Object	16#0400	Input Value Out of Range
			16#0419	Incorrect Data Type
			16#041B	Data Capacity Exceeded
			16#4800	Device Error Received
			16#4801	Specified Unit Does Not Exist
			16#4802	Message Processing Limit Exceeded
			16#4803	Specified Unit Status Error
			16#4804	Too Many Simultaneous Instruction Executions
			16#4805	Communications Timeout
			16#4806	Invalid Mode
			16#4807	I/O Power OFF Status
			16#4808	Verification Error
EtherNet/IP Communications Instructions	CIPOpen	Open CIP Class 3 Connection (Large_Forward_Open)	16#0400	Input Value Out of Range
			16#1C00	Explicit Message Error
			16#1C01	Incorrect Route Path
			16#1C03	CIP Communications Resource Overflow
			16#1C04	CIP Timeout
			16#1C05	Class-3 Connection Not Established
			16#2000	Local IP Address Setting Error
			16#2004	Local IP Address Not Set

Type	Instruction	Name	Error code	Error name
EtherNet/IP Communications Instructions	CIPOpenWithDataSize	Open CIP Class 3 Connection with Specified Data Size	16#0400	Input Value Out of Range
			16#1C00	Explicit Message Error
			16#1C01	Incorrect Route Path
			16#1C03	CIP Communications Resource Overflow
			16#1C04	CIP Timeout
			16#1C05	Class-3 Connection Not Established
			16#2000	Local IP Address Set-ting Error
			16#2004	Local IP Address Not Set
	CIPRead	Read Variable Class 3 Explicit	16#0400	Input Value Out of Range
			16#0407	Data Range Exceeded
			16#0419	Incorrect Data Type
			16#1C00	Explicit Message Error
			16#1C02	CIP Handle Out of Range
			16#1C03	CIP Communications Resource Overflow
			16#1C04	CIP Timeout
			16#1C06	CIP Communications Data Size Exceeded
	CIPWrite	Write Variable Class 3 Explicit	16#0400	Input Value Out of Range
			16#0406	Illegal Data Position Specified
			16#0407	Data Range Exceeded
			16#0419	Incorrect Data Type
			16#1C00	Explicit Message Error
			16#1C02	CIP Handle Out of Range
			16#1C03	CIP Communications Resource Overflow
			16#1C04	CIP Timeout
			16#1C06	CIP Communications Data Size Exceeded
	CIPSend	Send Explicit Message Class 3	16#0400	Input Value Out of Range
			16#0401	Input Mismatch
			16#0406	Illegal Data Position Specified
			16#0407	Data Range Exceeded
			16#0419	Incorrect Data Type
			16#1C00	Explicit Message Error
			16#1C02	CIP Handle Out of Range
			16#1C03	CIP Communications Resource Overflow
			16#1C04	CIP Timeout
			16#1C06	CIP Communications Data Size Exceeded
	CIPClose	Close CIP Class 3 Connection	16#1C02	CIP Handle Out of Range
			16#1C03	CIP Communications Resource Overflow

Type	Instruction	Name	Error code	Error name
EtherNet/IP Communications Instructions	CIPUCMMRead	Read Variable UCMM Explicit	16#0400	Input Value Out of Range
			16#0407	Data Range Exceeded
			16#0419	Incorrect Data Type
			16#1C00	Explicit Message Error
			16#1C01	Incorrect Route Path
			16#1C03	CIP Communications Resource Overflow
			16#1C04	CIP Timeout
			16#2000	Local IP Address Set-ting Error
			16#2004	Local IP Address Not Set
	CIPUCMMWrite	Write Variable UCMM Explicit	16#0400	Input Value Out of Range
			16#0406	Illegal Data Position Specified
			16#0419	Incorrect Data Type
			16#1C00	Explicit Message Error
			16#1C01	Incorrect Route Path
			16#1C03	CIP Communications Resource Overflow
			16#1C04	CIP Timeout
			16#2000	Local IP Address Set-ting Error
			16#2004	Local IP Address Not Set
	CIPUCMMSend	Send Explicit Message UCMM	16#0400	Input Value Out of Range
			16#0401	Input Mismatch
			16#0406	Illegal Data Position Specified
			16#0407	Data Range Exceeded
			16#0419	Incorrect Data Type
			16#1C00	Explicit Message Error
			16#1C01	Incorrect Route Path
			16#1C03	CIP Communications Resource Overflow
			16#1C04	CIP Timeout
			16#2000	Local IP Address Set-ting Error
			16#2004	Local IP Address Not Set
	SktUDPCreate	Create UDP Socket	16#0400	Input Value Out of Range
			16#2000	Local IP Address Set-ting Error
			16#2001	TCP/UDP Port Already in Use
			16#2003	Socket Status Error
			16#2004	Local IP Address Not Set
			16#2008	Socket Communica-tions Resource Over-flow

Type	Instruction	Name	Error code	Error name
EtherNet/IP Communications Instructions	SktUDPRcv	UDP Socket Receive	16#0400	Input Value Out of Range
			16#0407	Data Range Exceeded
			16#0419	Incorrect Data Type
			16#2003	Socket Status Error
			16#2006	Socket Timeout
			16#2007	Socket Handle Out of Range
			16#2008	Socket Communications Resource Overflow
	SktUDPSend	UDP Socket Send	16#0400	Input Value Out of Range
			16#0406	Data Range Exceeded
			16#0419	Incorrect Data Type
			16#2002	Address Resolution Failed
			16#2003	Socket Status Error
			16#2007	Socket Handle Out of Range
			16#2008	Socket Communications Resource Overflow
	SktTCPAccept	Accept TCP Socket	16#0400	Input Value Out of Range
			16#2000	Local IP Address Setting Error
			16#2001	TCP/UDP Port Already in Use
			16#2002	Address Resolution Failed
			16#2003	Socket Status Error
			16#2004	Local IP Address Not Set
			16#2006	Socket Timeout
			16#2008	Socket Communications Resource Overflow
	SktTCPConnect	Connect TCP Socket	16#0400	Input Value Out of Range
			16#2000	Local IP Address Setting Error
			16#2001	TCP/UDP Port Already in Use
			16#2002	Address Resolution Failed
			16#2003	Socket Status Error
			16#2004	Local IP Address Not Set
			16#2006	Socket Timeout
			16#2008	Socket Communications Resource Overflow

Type	Instruction	Name	Error code	Error name
EtherNet/IP Communications Instructions	SktTCPRcv	TCP Socket Receive	16#0400	Input Value Out of Range
			16#0407	Data Range Exceeded
			16#0419	Incorrect Data Type
			16#2003	Socket Status Error
			16#2006	Socket Timeout
			16#2007	Socket Handle Out of Range
			16#2008	Socket Communications Resource Overflow
	SktTCPSend	TCP Socket Send	16#0400	Input Value Out of Range
			16#0406	Data Range Exceeded
			16#0419	Incorrect Data Type
			16#2003	Socket Status Error
			16#2006	Socket Timeout
			16#2007	Socket Handle Out of Range
			16#2008	Socket Communications Resource Overflow
	SktGetTCPStatus	Read TCP Socket Status	16#2003	Socket Status Error
			16#2007	Socket Handle Out of Range
			16#2008	Socket Communications Resource Overflow
	SktClose	Close TCP/UDP Socket	16#2007	Socket Handle Out of Range
			16#2008	Socket Communications Resource Overflow
	SktClearBuf	Clear TCP/UDP Socket Receive Buffer	16#2007	Socket Handle Out of Range
			16#2008	Socket Communications Resource Overflow
	SktSetOption	Set TCP Socket Option	16#0400	Input Value Out of Range
			16#0419	Incorrect Data Type
			16#2003	Socket Status Error
			16#2007	Socket Handle Out of Range
			16#2008	Socket Communications Resource Overflow
	ChangeIPAdr	Change IP Address	16#0400	Input Value Out of Range
			16#040D	Illegal Unit Specified
			16#2400	No Execution Right
	ChangeFTPAccount	Change FTP Account	16#0400	Input Value Out of Range
			16#040D	Illegal Unit Specified
			16#2400	No Execution Right

Type	Instruction	Name	Error code	Error name
EtherNet/IP Communications Instructions	FTPGetFileList	Get FTP Server File List	16#0400	Input Value Out of Range
			16#2403	FTP Client Execution Limit Exceeded
			16#2405	Directory Does Not Exist (FTP)
			16#2406	FTP Server Connection Error
			16#2407	Destination FTP Server Execution Failure
	FTPGetFile	Get File from FTP Server	16#0400	Input Value Out of Range
			16#2403	FTP Client Execution Limit Exceeded
			16#2404	File Number Limit Exceeded
			16#2405	Directory Does Not Exist (FTP)
			16#2406	FTP Server Connection Error
			16#2407	Destination FTP Server Execution Failure
			16#2408	SD Memory Card Access Failed for FTP
			16#2409	Specified File Does Not Exist
			16#240A	Specified File Is Write Protected
			16#240C	Specified File Access Failed
	FTPPutFile	Put File onto FTP Server	16#0400	Input Value Out of Range
			16#2403	FTP Client Execution Limit Exceeded
			16#2404	File Number Limit Exceeded
			16#2405	Directory Does Not Exist (FTP)
			16#2406	FTP Server Connection Error
			16#2407	Destination FTP Server Execution Failure
			16#2408	SD Memory Card Access Failed for FTP
			16#2409	Specified File Does Not Exist
			16#240A	Specified File Is Write Protected
			16#240B	Failed To Delete Specified File
			16#240C	Specified File Access Failed

Type	Instruction	Name	Error code	Error name
EtherNet/IP Communications Instructions	FTPRemoveFile	Delete FTP Server File	16#0400	Input Value Out of Range
			16#2403	FTP Client Execution Limit Exceeded
			16#2404	File Number Limit Exceeded
			16#2405	Directory Does Not Exist (FTP)
			16#2406	FTP Server Connection Error
			16#2407	Destination FTP Server Execution Failure
			16#2409	Specified File Does Not Exist
	FTPRemoveDir	Delete FTP Server Directory	16#0400	Input Value Out of Range
			16#2405	Directory Does Not Exist (FTP)
			16#2406	FTP Server Connection Error
			16#2407	Destination FTP Server Execution Failure
Serial Communications Instructions	NX_SerialSend	Send No-protocol Data	16#0400	Input Value Out of Range
			16#0406	Illegal Data Position Specified
			16#040D	Illegal Unit Specified
			16#0419	Incorrect Data Type
			16#041D	Too Many Simultaneous Instruction Executions
			16#0C04	Multi-execution of Ports
			16#0C0C	Instruction Executed to Inapplicable Port
			16#0C0D	DIF Unit Initialized
	NX_SerialRcv	Receive No-protocol Data	16#0400	Input Value Out of Range
			16#0406	Illegal Data Position Specified
			16#0407	Data Range Exceeded
			16#040D	Illegal Unit Specified
			16#0419	Incorrect Data Type
			16#041D	Too Many Simultaneous Instruction Executions
			16#0C03	Full Reception Buffer
			16#0C04	Multi-execution of Ports
			16#0C05	Parity Error
			16#0C06	Framing Error
			16#0C07	Overrun Error
			16#0C0B	Serial Communications Timeout
			16#0C0C	Instruction Executed to Inapplicable Port
			16#0C0D	DIF Unit Initialized

Type	Instruction	Name	Error code	Error name
Serial Communications Instructions	NX_ModbusRtuCmd	Send Modbus-RTU General Command	16#0400	Input Value Out of Range
			16#0406	Illegal Data Position Specified
			16#0407	Data Range Exceeded
			16#040D	Illegal Unit Specified
			16#0419	Incorrect Data Type
			16#041D	Too Many Simultaneous Instruction Executions
			16#0C03	Full Reception Buffer
			16#0C04	Multi-execution of Ports
			16#0C05	Parity Error
			16#0C06	Framing Error
			16#0C07	Overrun Error
			16#0C08	CRC Mismatch
			16#0C0B	Serial Communications Timeout
			16#0C0C	Instruction Executed to Inapplicable Port
			16#0C0D	DIF Unit Initialized
			16#0C10	Exceptional Modbus Response
			16#0C11	Invalid Modbus Response
	NX_ModbusRtuRead	Send Modbus-RTU Read Command	16#0400	Input Value Out of Range
			16#0406	Illegal Data Position Specified
			16#040D	Illegal Unit Specified
			16#0419	Incorrect Data Type
			16#041D	Too Many Simultaneous Instruction Executions
			16#0C03	Full Reception Buffer
			16#0C04	Multi-execution of Ports
			16#0C05	Parity Error
			16#0C06	Framing Error
			16#0C07	Overrun Error
			16#0C08	CRC Mismatch
			16#0C0B	Serial Communications Timeout
			16#0C0C	Instruction Executed to Inapplicable Port
			16#0C0D	DIF Unit Initialized
			16#0C10	Exceptional Modbus Response
			16#0C11	Invalid Modbus Response

Type	Instruction	Name	Error code	Error name
Serial Communications Instructions	NX_ModbusRtuWrite	Send Modbus-RTU Write Command	16#0400	Input Value Out of Range
			16#0406	Illegal Data Position Specified
			16#040D	Illegal Unit Specified
			16#0419	Incorrect Data Type
			16#041D	Too Many Simultaneous Instruction Executions
			16#0C03	Full Reception Buffer
			16#0C04	Multi-execution of Ports
			16#0C05	Parity Error
			16#0C06	Framing Error
			16#0C07	Overrun Error
			16#0C08	CRC Mismatch
			16#0C0B	Serial Communications Timeout
			16#0C0C	Instruction Executed to Inapplicable Port
			16#0C0D	DIF Unit Initialized
			16#0C10	Exceptional Modbus Response
			16#0C11	Invalid Modbus Response
	NX_SerialSigCtl	Serial Control Signal ON/OFF Switching	16#0400	Input Value Out of Range
			16#040D	Illegal Unit Specified
			16#0419	Incorrect Data Type
			16#041D	Too Many Simultaneous Instruction Executions
			16#0C04	Multi-execution of Ports
			16#0C0B	Serial Communications Timeout
			16#0C0C	Instruction Executed to Inapplicable Port
			16#0C0D	DIF Unit Initialized
	NX_SerialBufClear	Clear Buffer	16#0400	Input Value Out of Range
			16#040D	Illegal Unit Specified
			16#0419	Incorrect Data Type
			16#041D	Too Many Simultaneous Instruction Executions
			16#0C04	Multi-execution of Ports
			16#0C0B	Serial Communications Timeout
			16#0C0C	Instruction Executed to Inapplicable Port
			16#0C0D	DIF Unit Initialized

Type	Instruction	Name	Error code	Error name
Serial Communications Instructions	NX_SerialStartMon	Start Serial Line Monitoring	16#0400	Input Value Out of Range
			16#040D	Illegal Unit Specified
			16#0419	Incorrect Data Type
			16#041D	Too Many Simultaneous Instruction Executions
			16#0C04	Multi-execution of Ports
			16#0C0B	Serial Communications Timeout
			16#0C0C	Instruction Executed to Inapplicable Port
			16#0C0D	DIF Unit Initialized
	NX_SerialStopMon	Stop Serial Line Monitoring	16#0400	Input Value Out of Range
			16#040D	Illegal Unit Specified
			16#0419	Incorrect Data Type
			16#041D	Too Many Simultaneous Instruction Executions
			16#0C04	Multi-execution of Ports
			16#0C0B	Serial Communications Timeout
			16#0C0C	Instruction Executed to Inapplicable Port
			16#0C0D	DIF Unit Initialized
SD Memory Card Instructions	FileWriteVar	Write Variable to File	16#0400	Input Value Out of Range
			16#1403	File Does Not Exist
			16#1405	File Already in Use
			16#1409	That File Name Already Exists
			16#140A	Write Access Denied
			16#140B	Too Many Files Open
			16#4400	Shared Folder Cannot Be Used
			16#4402	Shared Folder Insufficient Memory
			16#4404	Too Many Files/ Directories
			16#440D	File or Directory Name Is Too Long
			16#440E	Shared Folder Access Failed
	FileReadVar	Read Variable from File	16#0400	Input Value Out of Range
			16#1403	File Does Not Exist
			16#1405	File Already in Use
			16#140B	Too Many Files Open
			16#4400	Shared Folder Cannot Be Used
			16#440D	File or Directory Name Is Too Long
			16#440E	Shared Folder Access Failed

Type	Instruction	Name	Error code	Error name
SD Memory Card Instructions	FileOpen	Open File	16#0400	Input Value Out of Range
			16#1403	File Does Not Exist
			16#1405	File Already in Use
			16#140A	Write Access Denied
			16#140B	Too Many Files Open
			16#4400	Shared Folder Cannot Be Used
			16#4404	Too Many Files/ Directories
			16#440D	File or Directory Name Is Too Long
			16#440E	Shared Folder Access Failed
	FileClose	Close File	16#1403	File Does Not Exist
			16#1405	File Already in Use
			16#440E	Shared Folder Access Failed
	FileSeek	Seek File	16#1400	SD Memory Card Access Failure
			16#1403	File Does Not Exist
			16#1405	File Already in Use
			16#1407	Offset Out of Range
			16#4400	Shared Folder Cannot Be Used
			16#440E	Shared Folder Access Failed
	FileRead	Read File	16#0406	Illegal Data Position Specified
			16#0419	Incorrect Data Type
			16#1403	File Does Not Exist
			16#1405	File Already in Use
			16#1406	Open Mode Mismatch
			16#4400	Shared Folder Cannot Be Used
			16#440E	Shared Folder Access Failed
	FileWrite	Write File	16#0406	Illegal Data Position Specified
			16#0419	Incorrect Data Type
			16#1403	File Does Not Exist
			16#1405	File Already in Use
			16#1406	Open Mode Mismatch
			16#4400	Shared Folder Cannot Be Used
			16#4402	Shared Folder Insufficient Memory
			16#440E	Shared Folder Access Failed
	FileGets	Get Text String	16#1403	File Does Not Exist
			16#1405	File Already in Use
			16#1406	Open Mode Mismatch
			16#4400	Shared Folder Cannot Be Used
			16#440E	Shared Folder Access Failed

Type	Instruction	Name	Error code	Error name
SD Memory Card Instructions	FilePuts	Put Text String	16#1403	File Does Not Exist
			16#1405	File Already in Use
			16#1406	Open Mode Mismatch
			16#4400	Shared Folder Cannot Be Used
			16#4402	Shared Folder Insufficient Memory
			16#440E	Shared Folder Access Failed
	FileCopy	Copy File	16#0400	Input Value Out of Range
			16#1403	File Does Not Exist
			16#1405	File Already in Use
			16#1409	That File Name Already Exists
			16#140A	Write Access Denied
			16#140B	Too Many Files Open
			16#4400	Shared Folder Cannot Be Used
			16#4402	Shared Folder Insufficient Memory
			16#4404	Too Many Files/Directories
			16#440D	File or Directory Name Is Too Long
			16#440E	Shared Folder Access Failed
	FileRemove	Delete File	16#0400	Input Value Out of Range
			16#1403	File Does Not Exist
			16#1405	File Already in Use
			16#140A	Write Access Denied
			16#140B	Too Many Files Open
			16#4400	Shared Folder Cannot Be Used
			16#440D	File or Directory Name Is Too Long
			16#440E	Shared Folder Access Failed
	FileRename	Change File Name	16#0400	Input Value Out of Range
			16#1403	File Does Not Exist
			16#1405	File Already in Use
			16#1408	Directory Not Empty
			16#1409	That File Name Already Exists
			16#140A	Write Access Denied
			16#140B	Too Many Files Open
			16#4400	Shared Folder Cannot Be Used
			16#4404	Too Many Files/ Directories
			16#440D	File or Directory Name Is Too Long
			16#440E	Shared Folder Access Failed

Type	Instruction	Name	Error code	Error name
SD Memory Card Instructions	DirCreate	Create Directory	16#0400	Input Value Out of Range
			16#1405	File Already in Use
			16#1409	That File Name Already Exists
			16#140B	Too Many Files Open
			16#140C	Directory Does Not Exist
			16#4400	Shared Folder Cannot Be Used
			16#4402	Shared Folder Insuffi-cient Memory
			16#4404	Too Many Files/ Direc-tories
			16#440D	File or Directory Name Is Too Long
			16#440E	Shared Folder Access Failed
	DirRemove	Delete Directory	16#0400	Input Value Out of Range
			16#1405	File Already in Use
			16#1408	Directory Not Empty
			16#140A	Write Access Denied
			16#140B	Too Many Files Open
			16#140C	Directory Does Not Exist
			16#4400	Shared Folder Cannot Be Used
			16#440D	File or Directory Name Is Too Long
			16#440E	Shared Folder Access Failed
	BackupToMemoryCard	SD Memory Card Backup	16#0400	Input Value Out of Range
			16#1409	That File Name Already Exists
			16#140C	Directory Does Not Exist
			16#140F	Backup Operation Already in Progress
			16#1410	Cannot Execute Backup
			16#4400	Shared Folder Cannot Be Used
			16#4402	Shared Folder Insuffi-cient Memory
			16#4404	Too Many Files/ Direc-tories
			16#440E	Shared Folder Access Failed
			16#4411	Slave Backup Failed
OS Control Instructions	IPC_RebootOS	Restart OS	16#0400	Input Value Out of Range
			16#4000	OS Timeout
			16#4002	Error in Executing Restart OS

A-2 Instructions You Cannot Use in Event Tasks

An event task is executed only once when the specified execution condition is met. They are not executed repeatedly each task period. Therefore, programs that contain instructions that are executed over more than one task period cannot be assigned to event tasks.

The instructions in the following table are executed over more than one task. Do not use these instructions in programs that are assigned to an event task. If you do, a building error will occur.

Type	Instruction	Name	Page
Stack and Table Instructions	RecSort	Record Sort	2-524
Analog Control Instructions	PIDAT	PID Control with Autotuning	2-670
	PIDAT_HeatCool	Heating/Cooling PID with Autotuning	2-695
	AC_StepProgram	Step Program	2-777
System Control Instructions	ResetPLCError	Reset PLC Controller Error	2-823
	ResetMCError	Reset Motion Control Error	2-830
	ResetECError	Reset EtherCAT Error	2-837
	RestartNXUnit	Restart NX Unit	2-844
	NX_ChangeWriteMode	Change to NX Unit Write Mode	2-851
	NX_SaveParam	Save NX Unit Parameters	2-856
	NX_ReadTotalPowerOnTime	Read NX Unit Total Power ON Time	2-862
EtherCAT Communications Instructions	EC_CoESDOWrite	Write EtherCAT CoE SDO	2-908
	EC_CoESDORead	Read EtherCAT CoE SDO	2-911
	EC_StartMon	Start EtherCAT Packet Monitor	2-916
	EC_StopMon	Stop EtherCAT Packet Monitor	2-922
	EC_SaveMon	Save EtherCAT Packets	2-924
	EC_CopyMon	Transfer EtherCAT Packets	2-926
	EC_DisconnectSlave	Disconnect EtherCAT Slave	2-928
	EC_ConnectSlave	Connect EtherCAT Slave	2-935
	EC_ChangeEnableSetting	Enable/Disable EtherCAT Slave	2-937
	NX_WriteObj	Write NX Unit Object	2-954
	NX_ReadObj	Read NX Unit Object	2-969
IO-Link Communications Instructions	IOL_ReadObj	Read IO-Link Device Object	2-978
	IOL_WriteObj	Write IO-Link Device Object	2-987
EtherNet/IP Communications Instructions	CIPOpen	Open CIP Class 3 Connection (Large_Forward_Open)	2-998
	CIPOpenWithDataSize	Open CIP Class 3 Connection with Specified Data Size	2-1007
	CIPRead	Read Variable Class 3 Explicit	2-1011
	CIPWrite	Write Variable Class 3 Explicit	2-1017
	CIPSend	Send Explicit Message Class 3	2-1023
	CIPClose	Close CIP Class 3 Connection	2-1028
	CIPUCMMRead	Read Variable UCMM Explicit	2-1031
	CIPUCMMWrite	Write Variable UCMM Explicit	2-1036
	CIPUCMMSend	Send Explicit Message UCMM	2-1043
	SktUDPCreate	Create UDP Socket	2-1053
	SktUDPRcv	UDP Socket Receive	2-1061

Type	Instruction	Name	Page
EtherNet/IP Communications Instructions	SktUDPSend	UDP Socket Send	2-1064
	SktTCPAccept	Accept TCP Socket	2-1067
	SktTCPConnect	Connect TCP Socket	2-1070
	SktTCPRcv	TCP Socket Receive	2-1079
	SktTCPSend	TCP Socket Send	2-1082
	SktGetTCPStatus	Read TCP Socket Status	2-1085
	SktClose	Close TCP/UDP Socket	2-1088
	SktClearBuf	Clear TCP/UDP Socket Receive Buffer	2-1091
	SktSetOption	Set TCP Socket Option	2-1094
	ChangeIPAdr	Change IP Address	2-1099
	ChangeFTPAccount	Change FTP Account	2-1107
	FTPGetFileList	Get FTP Server File List	2-1111
	FTPGetFile	Get File from FTP Server	2-1128
	FTPPutFile	Put File onto FTP Server	2-1137
	FTPRemoveFile	Delete FTP Server File	2-1148
	FTPRemoveDir	Delete FTP Server Directory	2-1158
Serial Communications Instructions	NX_SerialSend	Send No-protocol Data	2-1164
	NX_SerialRcv	Receive No-protocol Data	2-1177
	NX_ModbusRtuCmd	Send Modbus RTU General Command	2-1191
	NX_ModbusRtuRead	Send Modbus RTU Read Command	2-1202
	NX_ModbusRtuWrite	Send Modbus RTU Write Command	2-1214
	NX_SerialSigCtl	Serial Control Signal ON/OFF Switching	2-1226
	NX_SerialBufClear	Clear Buffer	2-1235
	NX_SerialStartMon	Start Serial Line Monitoring	2-1245
	NX_SerialStopMon	Stop Serial Line Monitoring	2-1250
SD Memory Card Instructions	FileWriteVar	Write Variable to File	2-1256
	FileReadVar	Read Variable from File	2-1261
	FileOpen	Open File	2-1266
	FileClose	Close File	2-1270
	FileSeek	Seek File	2-1273
	FileRead	Read File	2-1277
	FileWrite	Write File	2-1285
	FileGets	Get Text String	2-1293
	FilePuts	Put Text String	2-1301
	FileCopy	Copy File	2-1310
	FileRemove	Delete File	2-1319
	FileRename	Change File Name	2-1324
	DirCreate	Create Directory	2-1329
	DirRemove	Delete Directory	2-1332
	BackupToMemoryCard	SD Memory Card Backup	2-1335
Time Stamp Instructions	NX_DOutTimeStamp	Write Digital Output with Specified Time Stamp	2-1352
	NX_AryDOutTimeStamp	Write Digital Output Array with Specified Time Stamp	2-1358

A-3 Instructions Related to NX Message Communications Errors

If too many of the following instructions are executed at the same time, an NX Message Communications Error may occur. If an NX Message Communications Error occurs, reduce the number of the following instructions that are executed. The conditions for an NX Message Communications Error depends on factors such as the communications traffic.

Classification	Instruction	Name	Page
System Control Instructions	RestartNXUnit	Restart NX Unit	2-844
	NX_ChangeWriteMode	Change to NX Unit Write Mode	2-851
	NX_SaveParam	Save NX Unit Parameters	2-856
	NX_ReadTotalPowerOnTime	Read NX Unit Total Power ON Time	2-862
EtherCAT Communications Instructions	EC_CoESDOWrite	Write EtherCAT CoE SDO	2-908
	EC_CoESDORead	Read EtherCAT CoE SDO	2-911
	EC_StartMon	Start EtherCAT Packet Monitor	2-916
	EC_StopMon	Stop EtherCAT Packet Monitor	2-922
	EC_SaveMon	Save EtherCAT Packets	2-924
	EC_CopyMon	Transfer EtherCAT Packets	2-926
	EC_DisconnectSlave	Disconnect EtherCAT Slave	2-928
	EC_ConnectSlave	Connect EtherCAT Slave	2-935
	EC_ChangeEnableSetting	Enable/Disable EtherCAT Slave	2-937
	NX_WriteObj	Write NX Unit Object	2-954
	NX_ReadObj	Read NX Unit Object	2-969
IO-Link Communications Instructions	IOL_ReadObj	Read IO-Link Device Object	2-978
	IOL_WriteObj	Write IO-Link Device Object	2-987

A-4 SDO Abort Codes

As reference information, the following table lists the SDO abort codes for EtherCAT communications. The abort codes that are used in actual communications are specified by the slaves. Refer to the slave manuals when programming communications.

Value	Meaning
16#05030000	Toggle bit not changed
16#05040000	SDO protocol timeout
16#05040001	Client/Server command specifier not valid or unknown
16#05040005	Out of memory
16#06010000	Unsupported access to an object
16#06010001	Attempt to read to a write only object
16#06010002	Attempt to write to a read only object
16#06020000	The object does not exist in the object directory
16#06040041	The object cannot be mapped into the PDO
16#06040042	The number and length of the objects to be mapped would exceed the PDO length
16#06040043	General parameter incompatibility reason
16#06040047	General internal incompatibility in the device
16#06060000	Access failed due to a hardware error
16#06070010	Data type does not match, length of service parameter does not match
16#06070012	Data type does not match, length of service parameter too high
16#06070013	Data type does not match, length of service parameter too low
16#06090011	Subindex does not exist
16#06090030	Value range of parameter exceeded (only for write access)
16#06090031	Value of parameter written too high
16#06090032	Value of parameter written too low
16#06090036	Maximum value is less than minimum value
16#08000000	General error
16#08000020	Data cannot be transferred or stored to the application
16#08000021	Data cannot be transferred or stored to the application because of local control*
16#08000022	Data cannot be transferred or stored to the application because of the present device state
16#08000023	Object dictionary dynamic generation failed or no object dictionary is present

* This is internal status that is unique to the slave.

Source: EtherCAT Specification Part 6 Application Layer Protocol Specification.

Document No.: ETG.1000.6 S (R) V1.0.2

A-5 Version Information

This appendix lists the instructions for which specifications were changed and instructions that were added for different unit versions of the Controllers and for different versions of the Sysmac Studio.

The instructions that are supported and their specifications depend on the unit version of the Controller and the version of the Sysmac Studio. These are given in the following table.

If a version is given for both the Controller and Sysmac Studio, both versions are required.

Type	Instruction	Name	New/Changed	Versions		Page
				CPU Unit	Sysmac Studio	
Math Instructions	EXPT (**)	Exponentiation	Changed	Ver. 1.16	Ver. 1.20	2-211
Conversion Instructions	LOW-ER_BOUND	Get First Number of Array	New	Ver. 1.18	Ver. 1.22	2-491
	UPPER_BOUND	Get Last Number of Array	New	Ver. 1.18	Ver. 1.22	2-491
EtherNet/IP Communications Instructions	FTPGetFileList	Get FTP Server File List	Changed	Ver. 1.16	---	2-1111
	FTPGetFile	Get File from FTP Server	Changed	Ver. 1.16	---	2-1128
	FTPPutFile	Put File onto FTP Server	Changed	Ver. 1.16	---	2-1137
	FTPRemove-File	Delete FTP Server File	Changed	Ver. 1.16	---	2-1148
	FTPRemove-Dir	Delete FTP Server Directory	Changed	Ver. 1.16	---	2-1158

I

Index

I

Index

Symbols

Numerics

A

B

C

D

I

O

P

R

I

S

T

U

W

X

Z

OMRON Corporation **Industrial Automation Company**
Kyoto, JAPAN

Contact: www.ia.omron.com

Regional Headquarters

OMRON EUROPE B.V.
Wegalaan 67-69, 2132 JD Hoofddorp
The Netherlands
Tel: (31)2356-81-300/Fax: (31)2356-81-388

OMRON ELECTRONICS LLC
2895 Greenspoint Parkway, Suite 200
Hoffman Estates, IL 60169 U.S.A.
Tel: (1) 847-843-7900/Fax: (1) 847-843-7787

OMRON ASIA PACIFIC PTE. LTD.
No. 438A Alexandra Road # 05-05/08 (Lobby 2),
Alexandra Technopark,
Singapore 119967
Tel: (65) 6835-3011/Fax: (65) 6835-2711

OMRON (CHINA) CO., LTD.
Room 2211, Bank of China Tower,
200 Yin Cheng Zhong Road,
PuDong New Area, Shanghai, 200120, China
Tel: (86) 21-5037-2222/Fax: (86) 21-5037-2200

Authorized Distributor:

© OMRON Corporation 2016-2019 All Rights Reserved.
In the interest of product improvement,
specifications are subject to change without notice.

Cat. No. W560-E1-06 0119